EC-Council

JAVA

C | ASE

Certified | Application Security Engineer



Certified Application Security Engineer

COURSEWARE

EC-COUNCIL OFFICIAL CURRICULA

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

Certified Application Security Engineer

@ErrorCatchers

EC-Council

Copyright © 2018 by EC-Council. All rights reserved. Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher, with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication. without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law. For permission requests, write to EC-Council, addressed "Attention: EC-Council," at the address below.

EC-Council New Mexico
101C Sun Ave NE
Albuquerque, NM 87109

Information has been obtained by EC-Council from sources believed to be reliable. EC-Council uses reasonable endeavors to ensure that the content is current and accurate, however, because of the possibility of human or mechanical error we do not guarantee the accuracy, adequacy, or completeness of any information and are not responsible for any errors or omissions or the accuracy of the results obtained from use of such information.

The courseware is a result of extensive research and contributions from subject matter experts from the field from all over the world. Due credits for all such contributions and references are given in the courseware in the research endnotes. We are committed towards protecting intellectual property. If you are a copyright owner (an exclusive licensee or their agent), and if you believe that any part of the courseware constitutes an infringement of copyright, or a breach of an agreed licence or contract, you may notify us at **legal@eccouncil.org**. In the event of a justified complaint, EC-Council will remove the material in question and make necessary rectifications.

The courseware may contain references to other information resources and security solutions, but such references should not be considered as an endorsement of or recommendation by EC-Council. All product and company names mentioned in this courseware are trademarks™ or registered® trademarks of their respective holders. Use of them does not imply any affiliation with or endorsement by them.

Readers are encouraged to report errors, omissions and inaccuracies to EC-Council at **legal@eccouncil.org**. If you have any issues, please contact **support@eccouncil.org**.

NOTICE TO THE READER

EC-Council does not warrant or guarantee any of the products, methodologies, or frameworks described herein or perform any independent analysis in connection with any of the product information contained herein. EC-Council does not assume, and expressly disclaims, any obligation to obtain and include information other than that provided to it by the manufacturer. The reader is expressly warned to consider and adopt all safety precautions that might be indicated by the activities described herein and to avoid all potential hazards. By following the instruction contained herein, the reader willingly assumes all risks in connection with such instructions. EC-Council makes no representations or warranties of any kind, including but not limited to, the warranties of fitness for particular purpose or merchantability, nor are any such representations implied with respect to the material set forth herein, and EC-Council takes no responsibility with respect to such material. EC-Council shall not be liable for any special, consequential, or exemplary damages resulting, in whole or in part, from the reader's use or reliance up this material.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Program Introduction

Today, businesses of all sizes are going online to stay relevant and competitive. Many of these organizations rely solely on their web applications/websites to deliver goods and services. These applications are increasingly serving critical business functions for the customers and performing sophisticated complex business tasks such as real-time sales, transactions, inventory management across multiple vendors including both B2B and B2C e-commerce, workflow and supply chain management, etc. However, these applications also pose huge business risks to the organizations as they hold and process a lot of secret information related to the businesses. The applications have become potent attack vectors for attackers. A successful application-level attack can help attackers in pilfering millions of dollars, affecting business continuity, damaging reputation or even resulting in complete business shutdown.

Organizations' traditional network infrastructure, security processes and platforms such as firewalls, IDS/IPS systems, etc. are ineffective in defending application-level attacks. According to GARTNER report, over 70% of security vulnerabilities exist at the application layer and not at the network or system layer. Implementing traditional security measures can give organizations only a false sense of security at the application level.

According to OWASP, "Software applications are designed and developed with functionality first in mind and security as a distant second or third." The only way to cope with the increasing concerns and business risks associated with insecure applications is to integrate security into the whole software development lifecycle. Coding securely alone does not fully ensure secure software development as a missed requirement or bad web design or insecure architecture can make the application vulnerable to different types of attacks. A comprehensive application security requires security vigilance at requirement gathering, design, implementation and deployment of application.

Certified Application Security Engineer (CASE) is a hands-on, instructor-led, comprehensive application security course, which encompasses security activities involved in all the phases of Software Development Lifecycle (SDLC). The course focuses on secure software development process in the real world. The course will help students gain in-depth knowledge and skills in gathering security requirements, designing secure application architecture, implementing secure coding practices, performing static application security testing (SAST) and dynamic application testing (DAST), and secure deployment.

The CASE program briefs on various application security issues that may arise due to incorrect security strategies incorporated during traditional software development lifecycle and their impact on the businesses. The program also focuses on selecting and implementing the right security strategies, guidelines, and practices that are widely accepted and applicable to most environments used today.

Please visit https://www.eccouncil.org for more information.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

About EC-Council

The International Council of Electronic Commerce Consultants, better known as EC-Council, was founded in late 2001 to address the need for well-educated and certified information security and e-business practitioners. EC-Council is a global, member-based organization composed of industry and subject matter experts all working together to set the standards and raise the bar in information security certification and education.

EC-Council first developed the *Certified Ethical Hacker* (C|EH) program with the goal of teaching the methodologies, tools, and techniques used by hackers. Leveraging the collective knowledge from hundreds of subject matter experts, the CEH program has rapidly gained popularity around the globe and is now delivered in more than 145 countries by more than 950 authorized training centers. More than 200,000 information security practitioners have been trained.

CEH is the benchmark for many government entities and major corporations around the world. Shortly after CEH was launched, EC-Council developed the *Certified Security Analyst* (ECSA). The goal of the ECSA program is to teach ground-breaking analysis methods that must be applied while conducting advanced penetration testing. The ECSA program leads to the *Licensed Penetration Tester* (LPT). The *Computer Hacking Forensic Investigator* (CHFI) was formed with the same design methodologies and has become a global standard in certification for computer forensics. EC-Council, through its impressive network of professionals and huge industry following, has developed various other programs in information security and e-business. EC-Council certifications are viewed as the essential certifications needed when standard configuration and security policy courses fall short. Providing a true, hands-on, tactical approach to security, individuals armed with the knowledge disseminated by EC-Council programs are security networks around the world and beating hackers at their own game.

Other EC-Council Programs

Security Awareness: Certified Secure Computer User

C|SCU
Certified Secure Computer User

The purpose of the CSCU training program is to provide students with the necessary knowledge and skills to protect their information assets. This class will immerse students into an interactive environment where they will acquire fundamental understanding of various computer and network security threats such as identity theft, credit card fraud, online banking phishing scams, virus and backdoors, emails hoaxes, sex offenders lurking online, loss of confidential information, hacking attacks and social engineering. More importantly, the skills learnt from the class helps students take the necessary steps to mitigate their security exposure.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Network Defense: Certified Network Defender

Students enrolled in the Certified Network Defender course, will gain a detailed understanding and hands on ability to function in real life situations involving network defense. They will gain the technical depth required to actively design a secure network in your organization. This course gives you the fundamental understanding of the true construct of data transfer, network technologies, software technologies so that you understand how networks operate, understand what software is automating and how to analyze the subject material.

You will learn how to protect, detect and respond to the network attacks. You will learn network defense fundamentals, the application of network security controls, protocols, perimeter appliances, secure IDS, VPN and firewall configuration. You will then learn the intricacies of network traffic signature, analysis and vulnerability scanning which will help you when you design greater network security policies and successful incident response plans. These skills will help you foster resiliency and continuity of operations during attacks.

Ethical Hacking: Certified Ethical Hacker

™ The Certied Ethical Hacker (CEH) program is the core of the most desired information security training system any information security professional will ever want to be in. The CEH, is the rst part of a 3 part EC-Council Information Security Track which helps you master hacking technologies. You will become a hacker, but an ethical one!

As the security mindset in any organization must not be limited to the silos of a certain vendor, technologies or pieces of equipment, This course was desgined to provide you with the tools and techniques used by hackers and information security professionals alike to break into an organization. As we put it, “To beat a hacker, you need to think like a hacker”.

This program will immerse you into the Hacker Mindset so that you will be able to defend against future attacks. It puts you in the driver’s seat of a hands-on environment with a systematic ethical hacking process.

Here, you will be exposed to an entirely di-erent way of achieving optimal information security posture in their organization; by hacking it! You will scan, test, hack and secure your own systems. You will be thought the Five Phases of Ethical Hacking and thought how you can approach your target and succeed at breaking in every time! The ve phases include Reconnaissance, Gaining Access, Enumeration, Maintaining Access, and covering your tracks.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Penetration Testing: EC-Council Certified Security Analyst

EC Council Certified Security Analyst, ECSA complements the Certified Ethical Hacker, CEH certification by exploring the analytical phase of ethical hacking. While the Certified Ethical Hacker certification exposes the learner to hacking tools and technologies, the Certified Security Analyst course takes it a step further by exploring how to analyze the outcome from these tools and technologies. Through groundbreaking penetration testing methods and techniques, this pen testing security training course helps students perform the intensive assessments required to effectively identify and mitigate risks to the security of the infrastructure.

This makes the Certified Security Analyst "Penetration Training" a relevant milestone toward achieving EC Council's Licensed penetration Tester, which also ingrains the learner in the business aspect of penetration testing. The Licensed Penetration Tester standardizes the knowledge base for penetration testing professionals by incorporating the best practices followed by experienced experts in the field. The objective of Certified Security Analyst is to add value to experienced security professionals by providing security training that will help them analyze the outcomes of their Vulnerability Assessments. Penetration Testing Training leads the learner into the advanced stages of ethical hacking.

Penetration Testing: Licensed Penetration Tester (Master)

The LPT (Master) is the world's first fully online, remotely proctored LPT (Master) practical exam, which challenges the candidates through a grueling 18 hours of performance based, hands-on exam categorized into three practical exams for six-hour duration each, which will test your perseverance and focus by forcing you to outdo yourself with each new challenge. The exam requires the candidates to demonstrate a methodical approach to test and validate security defenses. The LPT (Master) exam is developed with close collaboration with SMEs and practitioners around the world after a thorough job role, job task, and skills-gap analysis.

Computer Forensics: Computer Hacking Forensic Investigator

CHFI is a comprehensive course covering major forensic investigation scenarios that enables students to acquire necessary hands-on experience regarding various forensic investigation techniques. Learn how to utilize standard forensic tools to successfully carryout a computer forensic investigation, which can better prepare you to aid in the prosecution of perpetrators.

EC-Council's CHFI certifies individuals in the specific security discipline of computer forensics from a vendor-neutral perspective. The CHFI certification will fortify the application knowledge of law enforcement personnel, system administrators, security officers, defense and military

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

personnel, legal professionals, bankers, security professionals, and anyone who is concerned about the integrity of the network infrastructure.

Incident Handling: EC-Council Certified Incident Handler



The EC-Council Certified Incident Handler program is designed to provide the fundamental skills to handle and respond to the computer security incidents in an information system. The course addresses various underlying principles and techniques for detecting and responding to current and emerging computer security threats. Students will learn how to handle various types of incidents, risk assessment methodologies, and various laws and policy related to incident handling. After attending the course, they will be able to create incident handling and response policies and deal with various types of computer security incidents. Network Security Administration: EC-Council Certified Network Administrator.

Management: Certified Chief Information Security Officer



The Certified Chief Information Security Officer (CCISO) program was developed by EC-Council to fill a knowledge gap that existed in the information security industry. Most information security certifications focus on specific tools or practitioner capabilities. At the time that the CCISO program was developed, no certification existed to recognize the knowledge, skills, and aptitudes required for an experienced information security professional to perform the duties of an effective, competent CISO. In fact, at the time this program was developed, many questions existed about what a CISO really is and the value that this role contributes to an organization.

The CCISO Body of Knowledge helps to define the role of the CISO and to answer the questions about the contributions this person makes in an organization. EC-Council enhances this information through training opportunities conducted as instructor-led or self-study modules to ensure candidates have a complete understanding of the role. Then, EC-Council evaluates the knowledge of CCISO candidates with a rigorous exam that tests competence across five domains with which a seasoned security leader should be familiar.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

CASE Exam Information

CASE Exam Details	
Exam Title	**Certified Application Security Engineer (CASE) - JAVA**
Exam Code	**312-96**
Availability	**EC-Council Exam Portal** (please visit **https://www.eccexam.com**)
Duration	**2 Hours**
Questions	**50**
Passing Score	Please refer **https://cert.eccouncil.org/faq.html**

Please visit **https://cert.eccouncil.org** for more information.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Table of Contents

Module 01: Understanding Application Security, Threats, and Attacks....01

Module Objectives 3

What is a Secure Application 3

Need for Application Security 4

Most Common Application Level Attacks 6

- SQL Injection Attacks 7
- Cross-site Scripting (XSS) Attacks 7
- Parameter Tampering 8
- Directory Traversal 9
- Cross-site Request Forgery (CSRF) Attack 9
- Denial-of-Service (DoS) Attack 10

Session Attacks 11

Why Applications become Vulnerable to Attacks 13

Common Reasons for Existence of Application Vulnerabilities 13

- Common Flaws Existed due to Insecure Coding Techniques 14

Improper Input Validation 14

Insufficient Transport Layer Protection 15

- Improper Error Handling 15
- Insecure Cryptographic Storage 16
- Broken Authentication and Session Management 16

Unvalidated Redirects and Forwards 17

Insecure Direct Object References 17

- Failure to Restrict URL Access 18

What Constitutes a Comprehensive Application Security? 18

Application Security Frame 19

- 3W's in Application Security 20

Insecure Application: A Software Development Problem 21

Solution: Integrating Security in Software Development Life Cycle (SDLC) 21

Functional vs Security Activities in SDLC 22

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Advantages of Integrating Security in SDLC 22
Software Security Standards, Models, and Frameworks 23
The Open Web Application Security Project (OWASP) 23
OWASP TOP 10 Attacks-2017 24
The Web Application Security Consortium (WASC) 24
WASC Threat Classification 25
Software Security Framework 25
BSIMM vs OpenSAMM 26
Module Summary 27

Module 02: Security Requirements Gathering 29

Module Objectives 31
Importance of Gathering Security Requirements 31
Security Requirements 32
Gathering Security Requirements 32
Why We Need Different Approach for Security Requirements Gathering 33
Key Benefits of Addressing Security at Requirement Phase 33
Stakeholders Involvement in Security Requirements Gathering 34
Characteristics of Good Security Requirement: SMART 34
Types of Security Requirements 35
Security Requirement Engineering (SRE) 36
SRE Phases 36
Common Mistakes Made in Each Phase of SRE 38
Different Security Requirement Engineering Approaches/Model 39
Abuse Case and Security Use Case Modeling 39
Abuse Cases 40
Threatens Relationship 40
Abuse Case Modeling Steps 41
Abuse Cases: Advantages and Disadvantages 41
Abuse Case Template 42
Security Use Cases 42
Security Use Cases are Abuse Case Driven 43
Modeling Steps for Security Use Cases 43

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Mitigates Relationship 44
Abuse Case vs Security Use Case 44
Security Use Case: Advantages and Disadvantages 45
Security Use Case Template 45
Security Use Case Guidelines 46
Effectiveness of Abuse and Security Case 51
Abuser and Security Stories 52
Textual Description Template: Abuser Stories and Security Stories 53
Effectiveness of Abuser and Security Stories 54
Abuser Stories: Advantages and Disadvantages 55
Security Quality Requirements Engineering (SQUARE) 55
SQUARE Effectiveness 56
SQUARE Process 57
SQUARE: Advantages and Disadvantages 58
Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) 59
OCTAVE Effectiveness 60
OCTAVE Steps 61
OCTAVE: Advantages and Disadvantages 61
Module Summary 62

Module 03: Secure Application Design and Architecture 63

Module Objectives 65
Relative Cost of Fixing Vulnerabilities at Different Phases of SDLC 66
Secure Application Design and Architecture 67
Goal of Secure Design Process 67
Secure Design Actions 68
Secure Design Principles 68
Threat Modeling 75
Threat Modeling Phases 77
Threat Modeling Process 77
Identify Security Objective 78
Create an Application Overview 79

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Draw the End-to-End Deployment Architecture 79
Identify Various User Roles 80
Identify Use Cases Scenarios 80
Identify Technologies 81
Identify Application Security Mechanisms 81
Decompose Application 82
Prepare and Document Threat Model Information 82
Identify the External Dependencies 83
Identify the Entry Points 84
Identify the Assets 85
Identify the Trust Levels 87
Define Trust Levels to Entry points 88
Define Trust Levels to Assets 88
Perform Application Modelling using Data Flow Diagrams (DFDs) 89
Determine the Threats: Identify the Goal of an Attacker and Create Threat Profile 91
Determine the Threats: Create a Security Profile 92
Identify the Threats 94
Determine Countermeasures and Mitigation Security Controls 95
Document the Threats 96
Rating the Threats 96
Secure Application Architecture 98
Design Secure Application Architecture 98
Module Summary 99

Module 04: Secure Coding Practices for Input Validation 101

Module Objectives 103
Input Validation Pattern 103
Validation and Security Issues 104
Impact of Invalid Data Input 104
Data Validation Techniques 105
Input Validation using Frameworks and APIs 106
Open Source Validation Framework for Java 107

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Servlet Filters 107
Validation Filters for Servlet 108
Data Validation using OWASP ESAPI 108
Data Validation: Struts Framework 109
Struts Validator 109
Struts Validation and Security 110
Data Validation using Struts Validator 110
Avoid Duplication of Validation Forms 111
Secure and Insecure Struts Validation Code 111
Struts Validator Class 112
Secure and Insecure Code for Struts Validator Class 112
Enable the Struts Validator 113
Secure and Insecure Struts Validator Code 113
Struts2 Framework Validator 114
Struts2 Framework: Built-in Data Validators 115
Struts2 Framework Annotation Based Validators 116
Struts2 Custom Validation: Workflow Interceptor 116
Struts2 Ajax Validation: jsonValidation Interceptor 117
Data Validation: Spring Framework 117
Spring Validator 118
Data Validation: Spring MVC Framework 118
Implementing Validator 119
JSR 380 Bean Validator API 119
Configuring JSR 380 120
Custom Validator Implementation in Spring 120
Spring Validation and Security 121
Input Validation Errors 121
Improper Sanitization of Untrusted Data 122
Improper Validation of Strings 123
Improper Logging of User Inputs 124
Improper Incorporation of Malicious Inputs into Format Strings 125
Inappropriate Use of Split Characters in Data Structures 125
Improper Validation of Non-Character Code Points 126

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Improper Use of String Modification 127
Improper Comparison of Locale-dependent Data 128
Common Secure Coding Practices 129
SQL Injection 129
Prepared Statement 130
Stored Procedures 130
Stored Procedure for Securing Input Validation 131
Cross-site Scripting (XSS) 132
Whitelisting vs Blacklisting 132
Regular Expressions 134
Character Encoding 135
Cross-site Scripting (XSS) Countermeasures 136
HTML Encoding 137
HTML Encoding using ESAPI Encoder 138
Cross-site Request Forgery (CSRF) 138
Directory Traversal 139
Parameter Manipulation and Countermeasures 141
Protecting Application from Log Injection Attack 142
XML Injection 142
Command Injection 143
LDAP Injection 143
XML External Entity Attack 144
Unrestricted File Upload Attack 146
Prevent Unrestricted File Upload: Validate File Extension 146
Injection Attacks Countermeasures 147
CAPTCHA 147
Best Practices for Input Validation 150
Module Summary 151

Module 05: Secure Coding Practices for Authentication and Authorization 153

Module Objectives 155
Introduction to Authentication 155

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Java Container Authentication 156
Authentication Mechanism Implementation 157
Types of Authentication 157
Declarative vs Programmatic Authentication 158
Declarative Security Implementation 158
Programmatic Security Implementation 159
Java EE Authentication Implementation Example 160
Basic Authentication 160
How to Implement Basic Authentication? 161
Form-based Authentication 161
Form-based Authentication Implementation 162
Implementing Kerberos-Based Authentication 162
Secured Kerberos Implementation 163
Client Certificate Authentication 164
Certificate Generation with Keytool 165
Implementing Encryption and Certificates in Client Application 167
Authentication Weaknesses and Prevention 167
Brute Force Attack 168
Web-based Enumeration Attack 168
Weak Password Attacks 169
Introduction to Authorization 169
JEE Based Authorization 171
Access Control Model 171
Discretionary Access Control (DAC) 174
Mandatory Access Control (MAC) 174
Role-based Access Control (RBAC) 175
Servlet Container 175
Authorizing Users by Servlets 176
EJB Authorization 177
EJB Authorization Controls 177
Declarative Security with EJBs 178
Programmatic Security with EJBs 179
Java Authentication and Authorization (JAAS) 180

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

JAAS Features 181
JAAS Architecture 181
Pluggable Authentication Module (PAM) Framework 182
JAAS Classes 182
JAAS Subject and Principal 183
Authentication in JAAS 184
Authorization in JAAS 185
Subject Methods doAs() and doAsPrivileged() 186
Impersonation in JAAS 187
JAAS Permissions 187
LoginContext in JAAS 188
Creating LoginContext 188
LoginContext Instantiation 189
JAAS Configuration 189
Locating JAAS Configuration File 190
JAAS CallbackHandler and Callbacks 190
Login to Standalone Application 191
JAAS Client 192
LoginModule Implementation in JAAS 192
Methods Associated with LoginModule 193
LoginModule Example 194
Phases in Login Process 194
Java EE Security 195
Java EE Application Architecture 196
Java EE Servers as Code Hosts 196
Declaring Roles 197
HTTP Authentication Schemes 198
Authorization Common Mistakes and Countermeasures 199
Common Mistakes 199
Authentication and Authorization in Spring Security Framework 201
Spring Security Framework 201
Spring Security Modules 202
Spring Authentication 203

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Storing Username and Password 203
Securing Authentication Provider 204
Implementing HTTP Basic Authentication 205
Form-based Authentication 205
Implementing Digest Authentication 206
Security Expressions 207
URL-based Authorization 208
JSP Page Content Authorization 208
JSP Page Content Authorization with Domain Object's ACL 209
Method Authorization 209
Configuring Anonymous Login 210
Logout Feature Configuration 210
Remember-Me Authentication 211
Integrating Spring Security with JAAS 211
Spring JAAS Implementation 212
Defensive Coding Practices against Broken Authentication and Authorization 213
Do Not Store Password in Java String Object 213
Avoid Cookie based Remember-Me Use Persistent Remember-Me 214
Implement Appropriate Session Timeout 214
Prevent Session Stealing by Securing SessionID Cookie 215
Secure Development Checklists: Broken Authentication and Session Management 216
Module Summary 216

Module 06: Secure Coding Practices for Cryptography 217

Module Objectives 219
Java Cryptography 219
Need for Java Cryptography 220
Java Security with Cryptography 220
Java Cryptography Architecture (JCA) 221
Java Cryptography Extension (JCE) 222
Encryption and Secret Keys 223
Attack Scenario: Inadequate/Weak Encryption 223

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Encryption: Symmetric and Asymmetric Key 224
Encryption/Decryption Implementation Methods 224
SecretKeys and KeyGenerator 225
Implementation Methods of KeyGenerator Class 225
Creating SecretKeys with KeyGenerator Class 226
Cipher Class 227
The Cipher Class 227
Implementation Methods of Cipher Class 228
Insecure Code for Cipher Class using DES Algorithm 228
Secure Code for Cipher Class using AES Algorithm 229
Digital Signatures 229
Attack Scenario: Man-in-the-Middle Attack 230
Digital Signatures 230
The Signature Class 231
Implementation Methods of Signature Class 231
The SignedObjects 232
Implementing Methods of SignedObjects 232
The SealedObjects 233
Implementation Methods of SealedObject 233
Insecure and Secure Code for Signed/Sealed Objects 234
Java XML Digital Signature 235
Secure Socket Layer (SSL) 238
Java Secure Socket Extension (JSSE) 239
SSL and Security: Example 1 239
SSL and Security: Example 2 240
JSSE and HTTPS 241
Insecure HTTP Server Code 242
Secure HTTP Server Code 243
Key Management 243
Attack Scenario: Poor Key Management 244
Keys and Certificates 244
Key Management System 245
KeyStore 245

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Implementation Method of KeyStore Class 246
KeyStore: Persistent Data Stores 246
Key Management Tool: KeyTool 247
Digital Certificates 248
Certification Authorities 249
Signing Jars 250
Signing JAR Tool: Jarsigner 251
Signed Code Sources 252
Insecure Code for Signed Code Sources 253
Secure Code for Signed Code Sources 254
Hashing 254
Hashing Algorithms 255
Securing Hashed Password with Salt 256
Implementing Hashing with Salt in Spring Security 257
Java Card Cryptography 258
Spring Security: Crypto Module 261
Crypto Module 261
Spring Security Crypto Module 262
Implementing BCryptPasswordEncoder() 263
Configuring BCryptPasswordEncoder() in Spring Security 263
JavaScript Object Signing and Encryption (JOSE) 264
Attacks against JWT, JWS and JWE 264
Implementing JWS using Jose4J 265
Implementing JWE using Jose4J 265
Implementing JWK using Jose4J 266
Dos and Don'ts in Java Cryptography 266
Dos and Don'ts 267
Best Practices for Java Cryptography 269
Module Summary 270

Module 07: Secure Coding Practices for Session Management 271

Module Objectives 273
Session Management 273

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Session Tracking 274
Session Tracking Methods 274
Session Management in Spring Security 278
Spring Session Management 279
Session Management using Spring Security 280
Restricting Concurrent Sessions per User using Spring Security 280
Controlling Session Timeout 281
Prevent using URL Parameters for Session Tracking 282
Prevent Session Fixation with Spring Security 282
Use SSL for Secure Connection 283
Session Vulnerabilities and their Mitigation Techniques 284
Session Vulnerabilities 284
Types of Session Hijacking Attacks 285
Countermeasures for Session Hijacking 286
Countermeasures for Session ID Protection 287
Best Practices and Guidelines for Secured Sessions Management 287
Best Coding Practices for Session Management 288
Checklist to Secure Credentials and Session IDs 291
Guidelines for Secured Session Management 291
Module Summary 292

Module 08: Secure Coding Practices for Error Handling 293

Module Objectives 295
Introduction to Exceptions 295
Exception and Error Handling 296
Example of an Exception 297
Handling Exceptions in Java 298
Exception Classes Hierarchy 299
Exceptions and Threats 300
Erroneous Exceptional Behaviors 300
Suppressing or Ignoring Checked Exceptions 301
Disclosing Sensitive Information 302
Logging Sensitive Data 303

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Restoring Objects to Prior State, if a Method Fails 304
Avoid using Statements that Suppress Exceptions 305
Prevent Access to Untrusted Code that Terminates JVM 306
Never Catch java.lang.NullPointerException 307
Never Allow methods to Throw RuntimeException, Exception, or Throwable 307
Never Throw Undeclared Checked Exceptions 308
Never Let Checked Exceptions Escape from Finally Block 309
Dos and Don'ts in Error Handling 310
Dos and Don'ts in Exception Handling 310
Avoid using Log Error and Throw exception at Same Time 313
Spring MVC Error Handling 313
Handling Controller Exceptions with @ExceptionHandler Annotation 314
Handling Controller Exceptions with HandlerExceptionResolver 315
Spring MVC: Global Exception Handling 315
Global Exception Handling: HandlerExceptionResolver 316
Mapping Custom Exceptions to Statuscode with @ResponseStatus 317
Configure Custom Error Page in Spring MVC 317
Exception Handling in Struts 2 318
Best Practices for Error Handling 319
Best Practices for Handling Exceptions in Java 320
Introduction to Logging 322
Logging in Java 323
Example for Logging Exceptions 323
Logging Levels 324
Logging using Log4j 324
Log4j and Java Logging API 325
Java Logging using Log4j 326
Secure Coding in Logging 326
Vulnerabilities in Logging 327
Logging: Vulnerable Code and Secure Code 328
Secured Practices in Logging 331
Module Summary 332

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Module 09: Static and Dynamic Application Security Testing (SAST & DAST) 333

Module Objectives 335
Static Application Security Testing 335
Static Application Security Testing (SAST) 336
Objectives of SAST 336
Why SAST 337
Skills required for SAST 337
What to look for in SAST 338
Common Vulnerabilities Identified Through SAST 338
Types of SAST 339
Where does Secure Code Review Fit in SDLC? 339
SAST Steps 340
SAST Activities- flow Chart 340
Recommendation for Effective SAST 341
SAST Deliverable 341
Automated Source Code Analysis 342
Selecting Static Analysis Tool 344
Manual Secure Code Review 345
Manual Secure Code Review for Most Common Vulnerabilities 345
Code Review for PCI DSS Compliance 346
Code Review for Blacklisting Validation Approach 347
Code Review for Client-side Validation Approach 347
Code Review for Non-parametrized SQL Query 348
Code Review for XSS Vulnerability 348
Code Review for Weak Password Authentication 349
Code Review for Hard-coded Passwords 349
Code Review for Empty Password in Connection String 350
Code Review for Insecure Basic Authentication 350
Code Review for Open Redirect 351
Code Review for Insecure LDAP Authentication 351
Code Review for Insecure Authorization Mechanism 352
Code Review for Weak Password Length 352

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for use of Weak Hashing Algorithm 353
Code Review for Use of Weak Random Number Generator 353
Code Review for Use of Insecure PBE Work Factor 354
Code Review for use of Weak Encryption Algorithm 354
Code Review for Use of Insufficient Encryption Key Size 355
Code Review for Unsafe Decoding 355
Code Review for Use of SSL 356
Code Review for OS Command Injection 356
Code Review for LDAP Injection 357
Code Review for XML Injection 357
Code Review for Unsafe use of request parameter to execute SQL Query 358
Code Review for Insecure File Upload 358
Code Review for Directory Traversal 359
Code Review for Sensitive Information Exposure 359
Code Review for Sensitive Information Leakage 360
Code Review for Generic Exception Throwing and Catching 360
Code Review for Cookies Vulnerable to Client-side Scripts Attacks 361
Code Review for Cookies Vulnerable to CSRF Attacks 361
Code Review for Enabling Directory Listing 362
Code Review: Check List Approach 362
Sample Checklist 363
SAST Finding 366
SAST Report 367
SAST Reporting 367
Dynamic Application Security Testing (DAST) 368
Types of DAST 369
SAST Vs DAST 370
Automated Application Vulnerability Scanning Tools 370
Web Application Security Scanners 371
Additional Web Application Vulnerability Scanners 372
Proxy-based Security Testing Tools 372
Burp Suite 373
OWASP Zed Attack Proxy (ZAP) 373

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Additional Proxy-based Security Testing Tools 374
Choosing Between SAST and DAST 374
Module Summary 375

Module 10: Secure Deployment and Maintenance 377

Module Objectives 379
Secure Deployment 379
Prior Deployment Activity 380
Check the Integrity of Application Package Before Deployment 380
Review the Deployment Guide Provided by the Software Vendor 380
Deployment Activities: Ensuring Security at Various Levels 381
Ensuring Security at Host Level 381
Check and Configure the Security of Machine Hosting Web Server, Application Server, Database Server and Network Devices 382
Physical Security 382
Host Level Security 383
Ensuring Security at Network Level 383
Network Level Security 384
Ensuring Security at Application Level 385
Web Application Firewall (WAF) 386
Ensuring Security at Web Container Level (Tomcat) 388
Install and Configure Tomcat Securely 388
Remove Server Banner 389
Start Tomcat with Security Manager 389
Configure Default Servlet Not to Serve Index Pages 390
Replace Default Error Page 390
Replace Default server.xml 391
Protect Shutdown Port 391
Restrict Access to Tomcat Manager Applications 392
Protecting Resources with Realms 392
Store Passwords as Digest 393
Do Not Run Tomcat as Root 393
Configure Restricted Datasets 394

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Session Handling using App Mode in Tomcat 394
Role Based Security 395
Securing Tomcat at Network level 396
Java Runtime Security Configurations 398
Tomcat General Security Setting 399
Verify Trace Element Setting in sever.xml 401
Verify CustomError Settings in web.xml 401
Verify maxPostSize Setting 402
Tomcat Security Checklist 402
Checklist for Security Configuration in server.xml File in Apache Tomcat 403
Tomcat High Availability 403
Best Practices for Securing Tomcat 405
Ensuring Security in Oracle 406
Oracle Database General Security Overview 406
Methods of Authentication in Oracle 407
Authentication by Oracle Database 407
Oracle Security Features 408
Default Database Installation and Configuration Security 409
Managing User Accounts Securely for the Site 410
Securing User Accounts 410
Password Management 412
Lock all Expired Accounts 412
Assign Users to Password Profile 413
Disable Remote Operating System Authentication 413
Securing Data 414
Restrict Access to Operating System Directories 414
Securing Database Installation and Configuration 415
Securing Network 415
How to Configure Encryption on the Client and the Server 416
Control Access Data 416
Virtual Private Database 417
Oracle Label Security 418
Database Vault 418

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Audit Vault 420
Built-in Audit Tools 420
Recommended Audit Settings 424
Security Maintenance and Monitoring 424
Post Deployment Activities: Security Maintenance and Monitoring 425
Security Maintenance Activities at OS Level 425
Security Maintenance Activities at Web Container Level 426
Security Maintenance Activities at Application Level 426
Module Summary 427
References 429

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

JAVA

C|ASE

Certified | Application Security Engineer

Module 01

Understanding Application Security, Threats, and Attacks

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

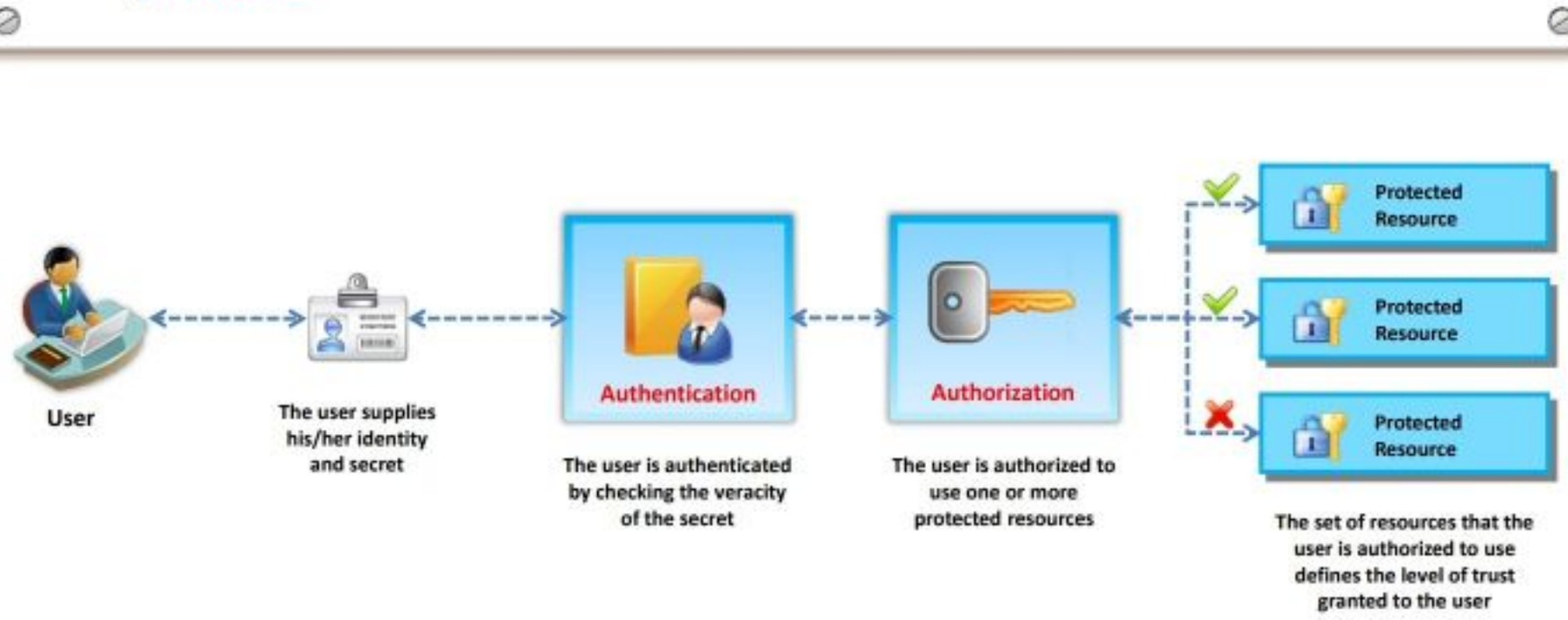
Module Objectives

1. Understand the Need and Benefits of Application Security
2. Discuss the Most Common Application-level Attacks
3. Discuss the Common Reasons Behind the Existence of Application-level Vulnerabilities
4. Understand Various Components of Comprehensive Application Security
5. Understand the Need and Advantages of Integrating Security in Software Development Life Cycle (SDLC)
6. Differentiate Functional vs Security Activities in SDLC
8. Discuss Various Software Security Reference Standards, Models, and Frameworks

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

What is a Secure Application

- An application is said to be secure when it ensures **confidentiality**, **integrity** and **availability** of its restricted resources
- Restricted resource is any **object**, **data**, **feature**, or **function** of an application designed to be accessed by only **authorized** users



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Need for Application Security

Organizations are increasingly using **web applications** to provide high value business functions to their customers such as real-time sales, transactions, inventory management across multiple vendors including both B-B and B-C e-commerce, workflow and supply chain management, etc.

Attackers exploit vulnerabilities in the applications to **launch various attacks** and **gain unauthorized access to resources**

A successful application level attack may result into:

1. Financial Loss
2. Affects Business Continuity
3. Closure of Business
4. Disclosure of Business Information
5. Damages Reputation
6. Fraudulent Transactions

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

"A vulnerability in an application will allow a malicious user to exploit a network or a host."

--Carlos Lyons, Corporate Security, Microsoft

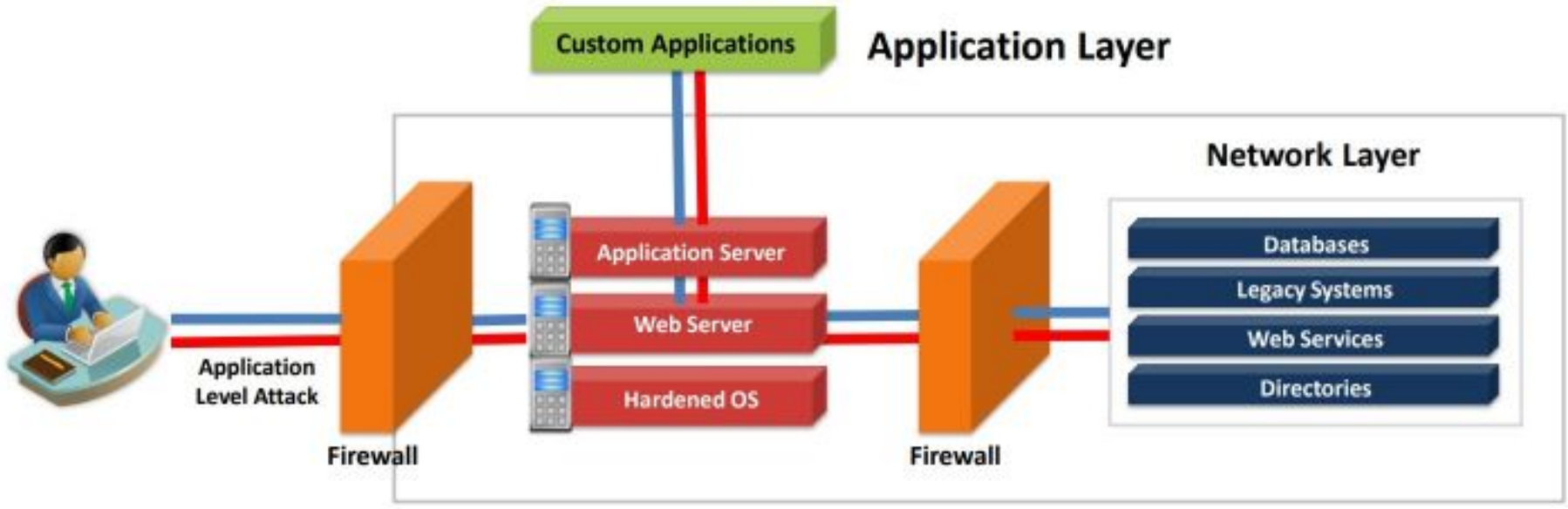
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Need for Application Security (Cont'd)

- It is a common myth that perimeter security controls such as firewall and IDS systems can secure your application but it is not true as these controls are **not effective** to defend application layer attacks
- This is because port 80 and 443 are generally open on perimeter devices for legitimate web traffic, which attackers can use to **exploit the application level vulnerabilities** and get into the network



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

"Nearly 75% of all attacks on information security are directed to the web application layer."

-- Gartner Report

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

"2/3 of all web applications are vulnerable."

-- Gartner Report

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Most Common Application Level Attacks

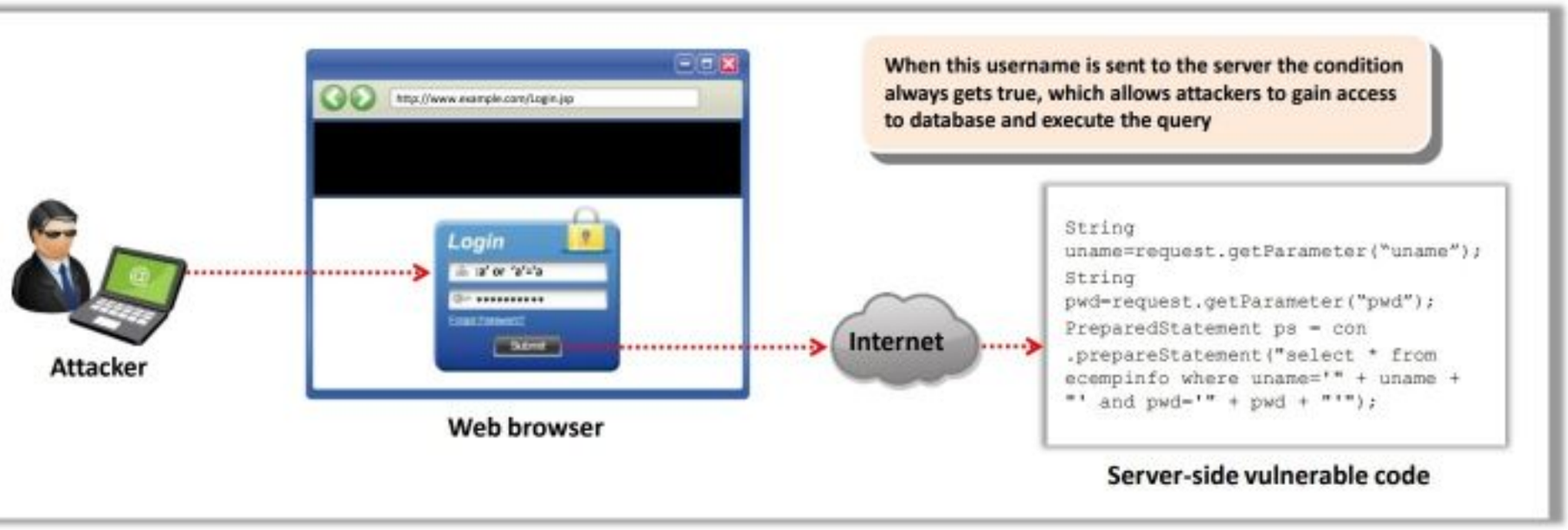
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

SQL Injection Attacks

- SQL injection attacks use a **series of malicious SQL queries** to directly manipulate the database
- An attacker can use a vulnerable web application to **bypass normal security measures** and obtain direct access to valuable data
- SQL injection attacks can often be executed from the **address bar**, from within application fields, and through queries and searches
- This attack is possible only when the application executes **dynamic SQL statements** and stores procedures with **arguments** based on the user input



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Cross-site Scripting (XSS) Attacks

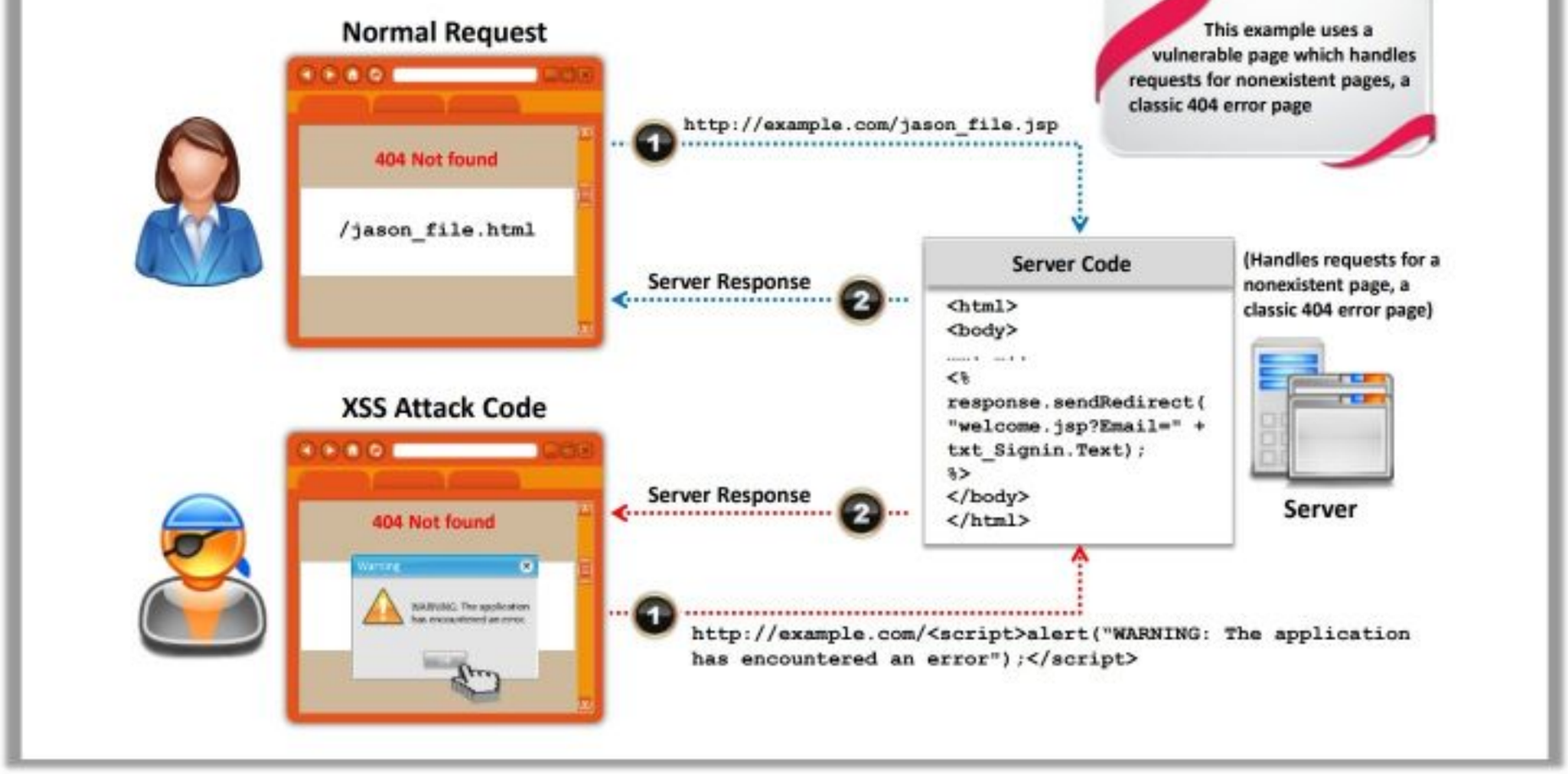
- Cross-site scripting ('XSS' or 'CSS') attacks **exploit vulnerabilities in dynamically generated web pages**, which enables malicious attackers to inject client-side script into web pages viewed by other users
- It occurs when **invalidated input data** is included in dynamic content that is sent to a user's web browser for rendering
- Attackers inject malicious JavaScript, VBScript, ActiveX, HTML, or Flash for execution on a victim's system by hiding it **within legitimate requests**

Malicious script execution	Session hijacking
Redirecting to a malicious server	Brute force password cracking
Exploiting user privileges	Data theft
Ads in hidden IFRAMES and pop-ups	Intranet probing
Data manipulation	Keylogging and remote monitoring

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

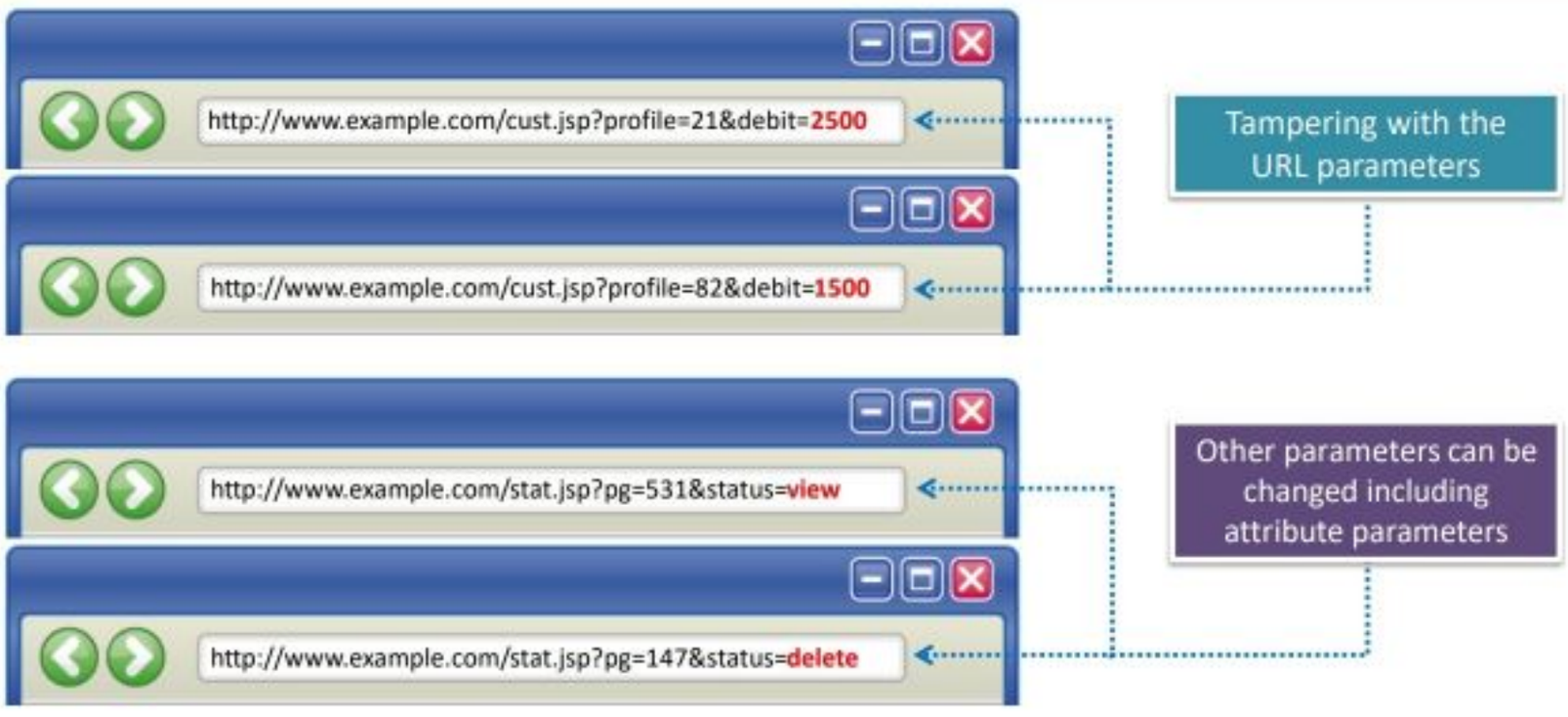
Cross-site Scripting (XSS) Attacks (Cont'd)



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Parameter Tampering

- A web parameter tampering attack involves **manipulation of parameters exchanged** between client and server in order to modify application data such as user credentials and permissions, price, and quantity of products
- A parameter tampering attack **exploits vulnerabilities** in integrity and logic validation mechanisms that may result in XSS, SQL injection, etc.



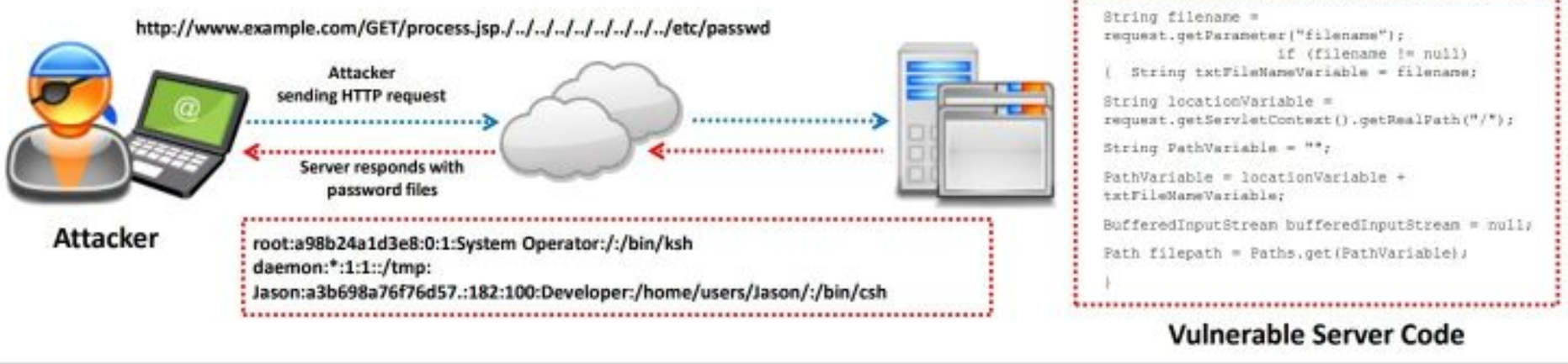
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Directory Traversal

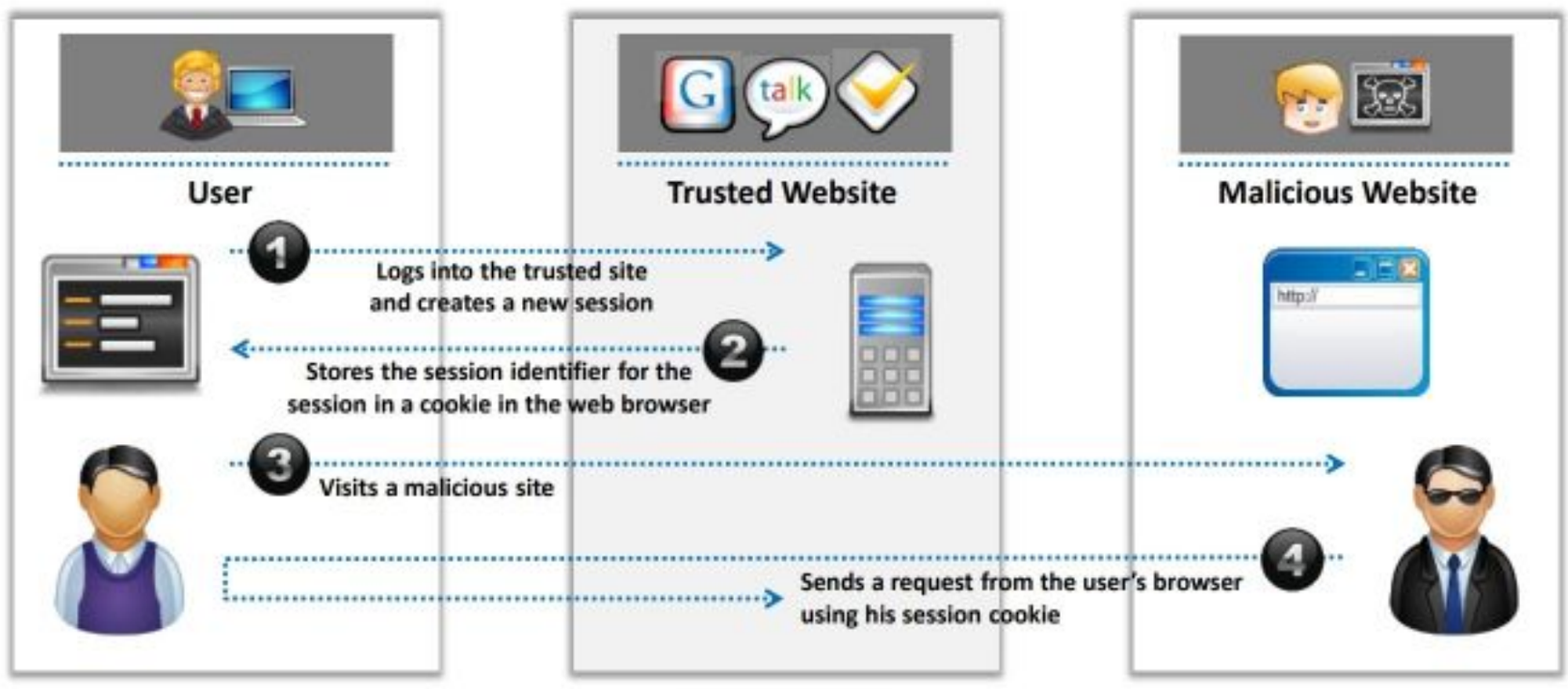
- Directory traversal allows attackers to **access restricted directories** including application source code, configuration, and critical system files and execute commands outside the webserver's root directory
- Accessing files located outside the **web publishing directory** using directory traversal
- Attackers can **manipulate variables** that reference files with "dot-dot-slash (../)" sequences and its variations
  - `http://www.example.com/process.jsp=../../../../some dir/some file`
  - `http://www.example.com/../../../../some dir/some file`



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Cross-site Request Forgery (CSRF) Attack

- Cross-site request forgery (CSRF) attacks **exploit web page vulnerabilities** that allow an attacker to force an unsuspecting user's browser to send malicious requests, which they did not intend to send
- The user, who is the victim **holds an active session** with a trusted site and simultaneously visits a malicious site, which **injects an HTTP request** for the trusted site into the victim user's session, compromising its integrity

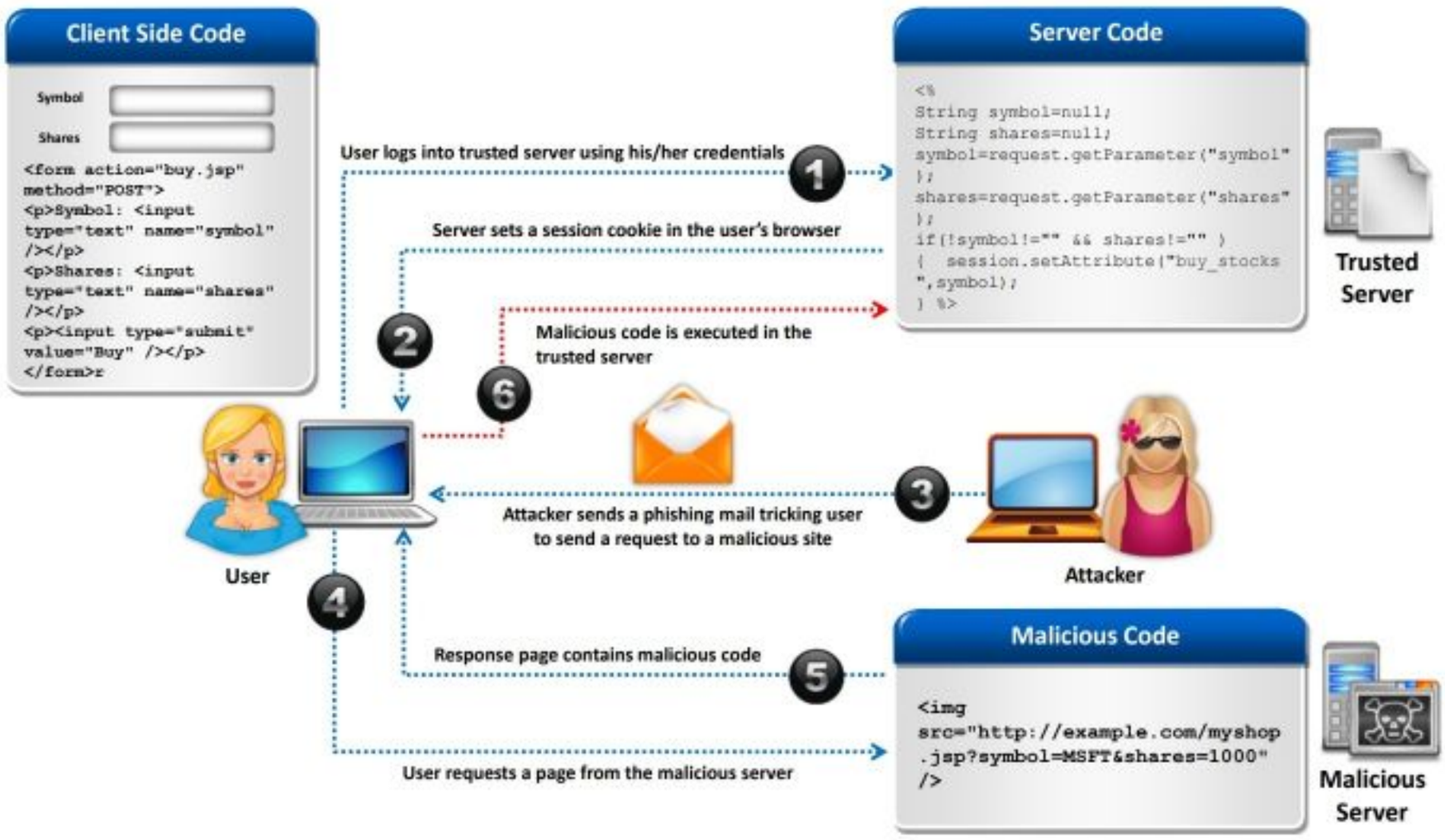


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Cross-site Request Forgery (CSRF) Attack (Cont'd)



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Denial-of-Service (DoS) Attack

- Attackers exhaust available server resources by sending hundreds of **resource-intensive requests** such as pulling out large image files or requesting dynamic pages that require expensive search operations on the backend of database servers
- Application-level DoS attacks emulate the same request syntax and network-level traffic characteristics as that of the legitimate clients, which makes it **undetectable by existing DoS protection** measures

Targets

- CPU, Memory, and Sockets
- Disk Bandwidth
- Database Bandwidth
- Worker Processes

Why are Applications Vulnerable to DoS?

- Reasonable Use of Expectations
- Application Environment Bottlenecks
- Implementation Flaws
- Poor Data Validation

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Denial-of-Service (DoS) Attack: Examples

User Registration DoS

- The attacker could create a program that submits the registration forms repeatedly; adding a **large number of spurious users** to the application

User Enumeration

- If **application states**, which are a part of the username/password pair is incorrect, an attacker can automate the process of trying **common usernames** from a **dictionary file** to enumerate the users of the application

Login Attacks

- The attacker may overload the login process by continually sending login requests that require the presentation tier to access the authentication mechanism, rendering it **unavailable** or **unreasonably** slow to respond

Account Lock Out Attacks

- The attacker may enumerate usernames through another vulnerability in the application and then attempt to authenticate the site using **valid usernames and incorrect passwords** which will lock out the accounts after a specified number of failed attempts. At this point legitimate users will not be able to use the site

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Session Attacks: Cookie Poisoning Attacks

- **Cookies are used to maintain session state in the otherwise stateless HTTP protocol**

Modify the Cookie Content

- Cookie poisoning attacks involve modification of the contents of a cookie (personal information stored in a web user's computer) in order to **bypass security mechanisms**

Inject the Malicious Content

- Poisoning allows an attacker to inject malicious content, modify the user's online experience, and obtain **unauthorized information**

Rewriting the Session Data

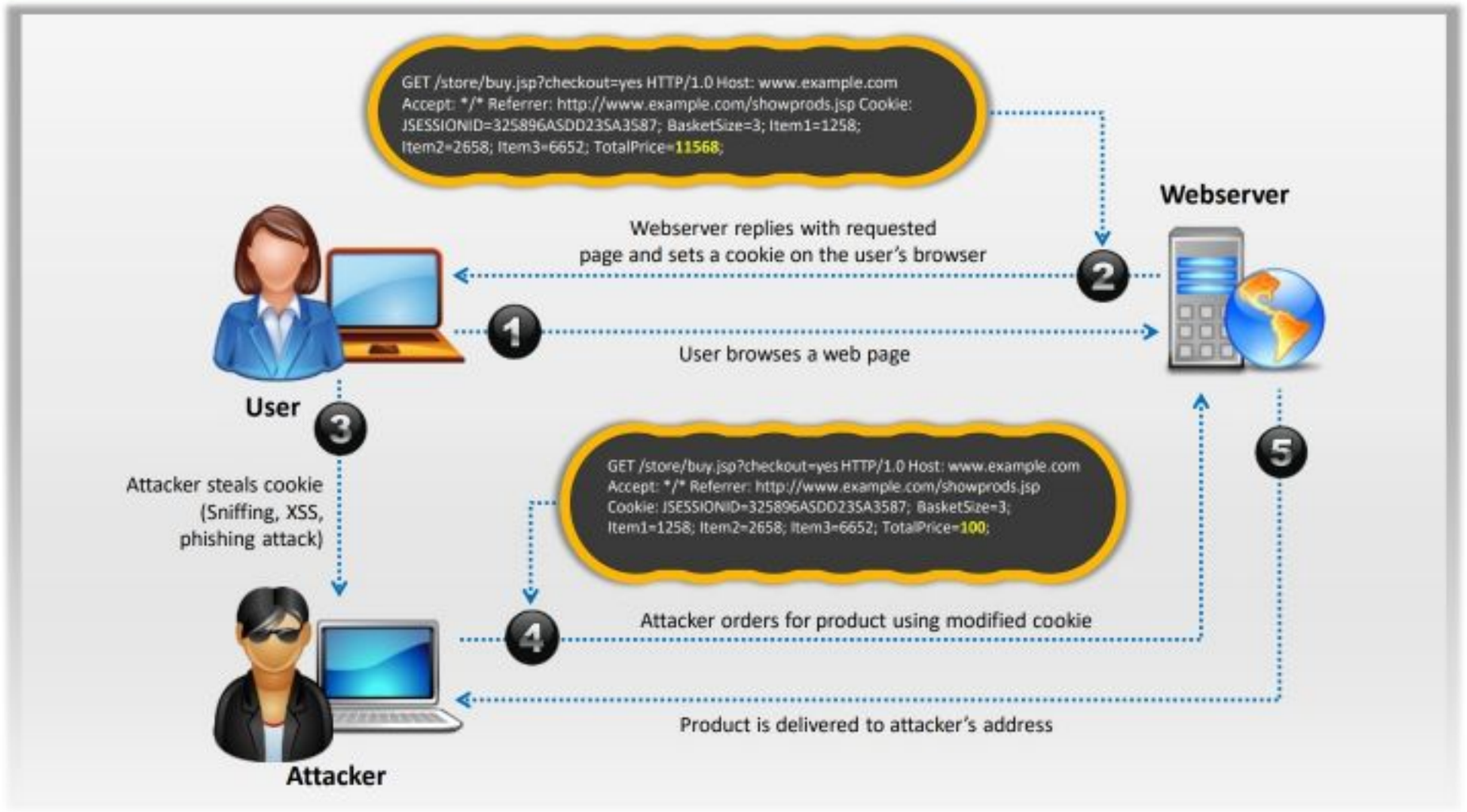
- A proxy can be used for rewriting the session data, displaying the cookie data, and/or specifying a new **user ID or other session identifiers** in the cookie

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

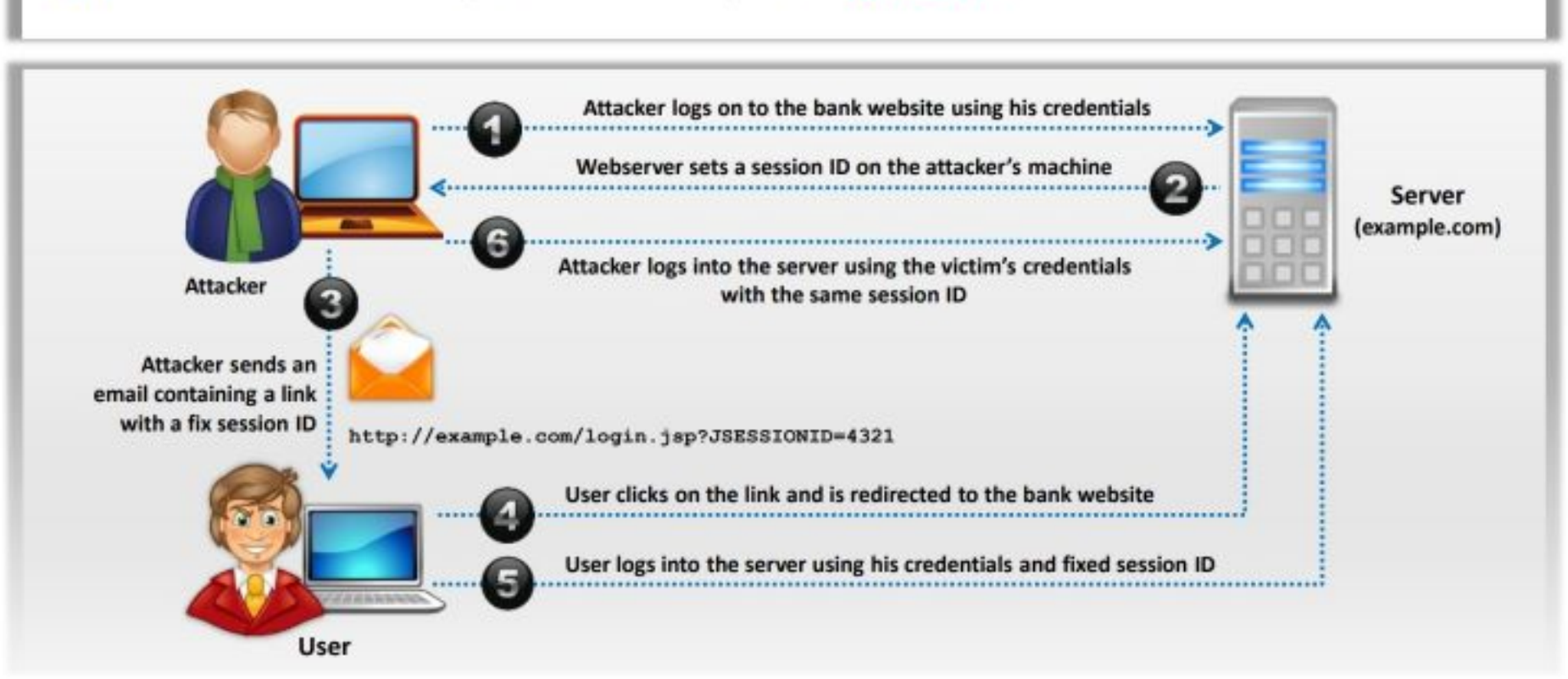
Session Attacks: Cookie Poisoning Attacks (Cont'd)



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Session Attacks: Session Fixation

- In a session fixation attack, the attacker tricks the user to access a genuine webserver using an **explicit session ID** value
- Attacker assumes the identity of the victim and exploits his **credentials** at the server



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Why Applications become Vulnerable to Attacks

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Common Reasons for Existence of Application Vulnerabilities

- Most of the software development related curriculum often **do not address security issues**
- No **proper guidance** provided to relevant stakeholders at different phases of the project development
- Failure to **gather** application security requirements in the **inception phase**
- Improper application of **security principles** in the **design phase**
- Insecure **coding techniques** give space to various vulnerabilities
- Lack of security **testing** in the **testing phase**
- Security **negligence** in the **deployment phase**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Common Flaws due to Insecure Coding Techniques

- Improper Input Validation
- Broken Authentication and Session Management
- Insecure Cryptographic Storage
- Improper Error Handling
- Insufficient Transport Layer Protection
- Unvalidated Redirects and Forwards
- Insecure Direct Object References
- Failure to Restrict URL Access

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Improper Input Validation

Invalidated Input Data

- The application receives inputs from various sources such as human users, software agents (browsers), and network/peripheral devices that can be **suspicious or untrusted**
- Processing inputs without proper validation can expose application to numerous threats

Malicious Script Execution

- Improper validation of input may provide path to the attackers to perform **injection attacks** such as cross site scripting attacks, SQL injection attacks, etc.
- The attacker can use various tricks and techniques to exploit the weaknesses in input validation mechanism of an application. They may **submit bogus data** to crash the system, maliciously manipulate the database, and corrupt the database of the application

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Insufficient Transport Layer Protection

Insufficient Transport Layer Protection

Exposes Data	This vulnerability exposes user's data to **untrusted third parties** and can lead to account theft
Launch Attacks	Underprivileged SSL setup can also help the attacker to launch phishing and **MITM attacks**
Supports Weak Algorithm	Insufficient transport layer protection supports weak algorithms, and uses **expired** or **invalid certificates**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Improper Error Handling

- Improper error handling **gives insight into source code** such as logic flaws, default accounts, etc.
- Using the information received from an error message, an attacker **identifies vulnerabilities** for launching various web application attacks

Information Gathered

- Out of memory
- Null pointer exceptions
- System call failure
- Database unavailable
- Network timeout
- Database information
- Web application logical flow
- Application environment



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Insecure Cryptographic Storage

- Insecure cryptographic storage refers to an **application** when it **uses poorly written encryption code** to encrypt and store sensitive data in the database

- This flaw allows an attacker to **steal or modify weakly protected data** such as credit card numbers, SSNs, and other authentication credentials

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Broken Authentication and Session Management

- An attacker uses vulnerabilities in the **authentication** or **session management functions** such as exposed accounts, session IDs, logout, password management, timeouts, remember me, secret question, account update, and others to impersonate users

Session ID in URLs

http://example.com/sale/saleitems=304;jsessionid=12OMTOIDPXM0OQSABGCKLHCJUN2JV?dest=NewMexico

- Attacker sniffs **the network traffic** or tricks the user to get the session IDs, and reuses the session IDs for malicious purposes

Password Exploitation

- Attacker gains access to the web application's **password database**. If user passwords are not encrypted, attackers can exploit **passwords of all the** users

Timeout Exploitation

- If an application's timeouts are not set properly and a user simply closes the browser without logging out from sites accessed through a public computer **then** the attacker can use the same browser later and **exploit** the **user's privileges**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Unvalidated Redirects and Forwards

- Unvalidated redirects enable attackers to **install malware or trick victims** into disclosing passwords or other sensitive information, whereas unsafe forwards may allow access control bypass





Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Insecure Direct Object References

- **Direct Object References**
  - Direct object reference vulnerability exists when a developer uses **internal object implementation** directly on the user interface
  - These objects may include **file**, **directory**, **database record**, or **key**
  - Attacker gets the idea about internal implementation and may use these objects as part of **URL** to access and modify sensitive information without any authorization
  - In bank application, attacker may use **account number** directly as an input field in the user interface, as he/she knows that account number is generally the primary key of any bank application
  - This may give path to the attacker to commit attacks on the **bank database**



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Failure to Restrict URL Access

- There are certain pages in the application which handles sensitive operations and data. Access to these pages are restricted, given based on authorization. For example, certain functionality of the application needs only **administrator rights**
- During development, the **developer** tries to hide these pages from others by not providing direct links to them
- The attacker uses **forceful browsing techniques** to find the URL of such sensitive pages to gain **unauthorized access**



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

What Constitutes a Comprehensive Application Security?

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

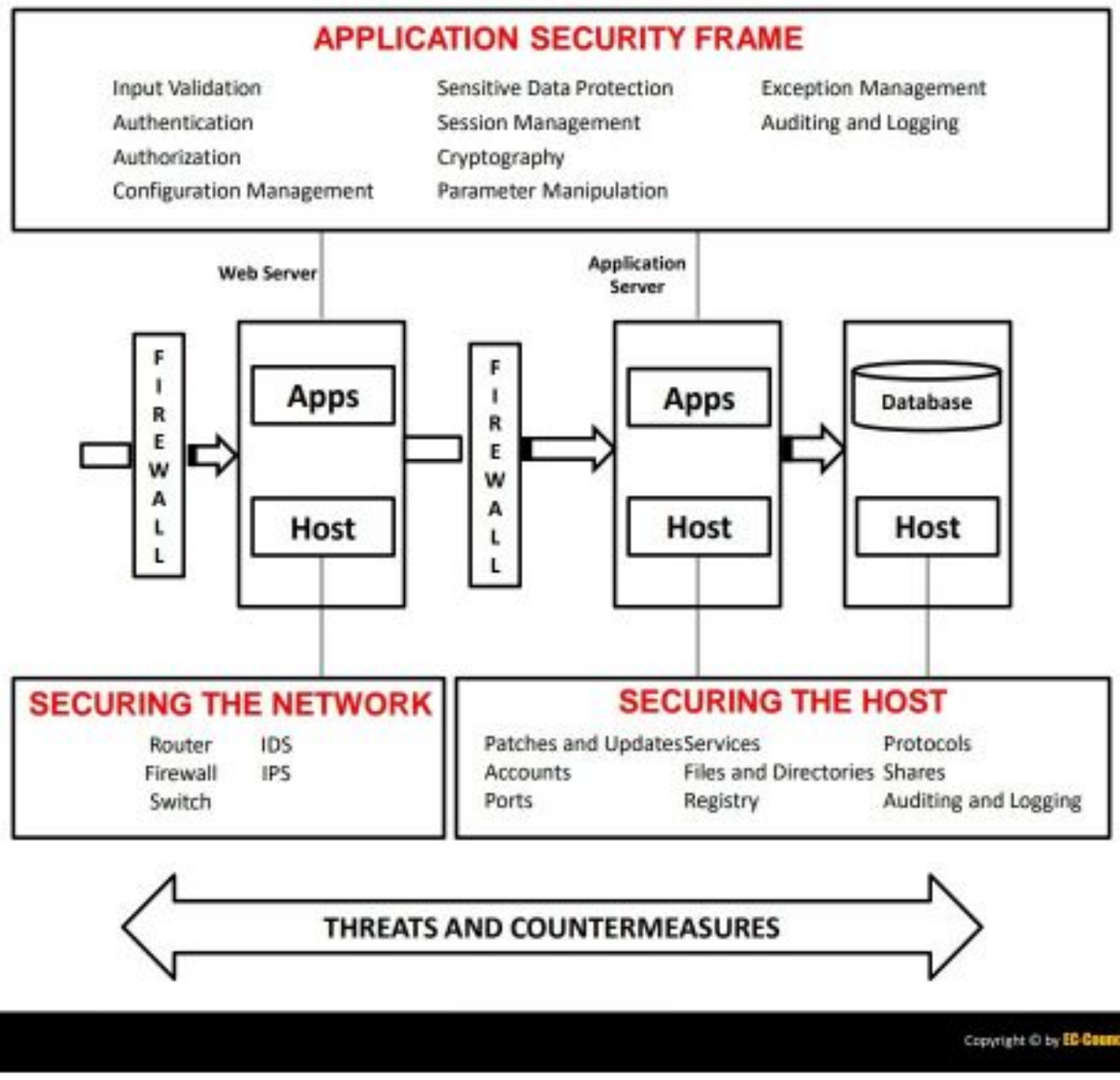
@ErrorCatchers

"Software application are designed and developed with functionality first in mind and security as a distant second or third."

-- OWASP

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Application Security Frame



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Coding securely alone does not ensure secure software development, a missed requirement or bad web design or insecure architecture can make application vulnerable to different types of attacks.

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

3W's in Application Security

Why	Why should we care about application security?
Due to its globally accessible nature, applications are becoming more popular targets for attackers to compromise an organization's security	

What	What do we need for application security ?
A constant security vigilance at various phase of the application development lifecycle	

Who	Who is responsible for application security?
Managers, Architects, Developers, Testers, and Administrators	

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Insecure Application: A Software Development Problem

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Solution: Integrating Security in Software Development Life Cycle (SDLC)

Security Software Development Process

Requirement	Design	Development	Testing	Deployment	Maintenance
Security Requirements	Security Requirements Secure Coding Standards Threat Modeling Security Architecture	Secure Coding Standards Secure Design Patterns and Frameworks Secure Coding Practices	Secure Code Review Vulnerability Assessment Penetration Testing	Secure Deployment	Security Patch Updates

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Functional vs Security Activities in SDLC

Software Development Lifecycle	Functional Activities	Security Activities
Requirement	• Functional requirements • Non functional requirements • Technology requirements	Defining security requirements
Design	Design guidelines and architectural design of project	• Secure design • Secure coding standards • Perform threat modeling • Secure architecture
Development	Programming functional logic, unit testing	• Implementing security requirements • Implementing secure coding standards • Adopting secure coding practices
Testing	Functional testing such as black box testing, white box testing etc.	Security testing
Deployment	Deployment	Secure deployment
Maintenance	Update functionality	Updating with security patches

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Advantages of Integrating Security in SDLC

- Reduce the presence of **software vulnerabilities** to great extent
- Ability to **comply with regulations**, standards or requirements for secure software development
- Reduce **costly rework** by detecting and eliminating flaws at the earliest phase
- Improves developer **job satisfaction**
- Improves **customer satisfaction**
- Embeds **security culture** to improve quality and reliability
- Reuse of **trusted software** in future development
- Reduce the **maintenance cost**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Software Security Standards, Models, and Frameworks

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

The Open Web Application Security Project (OWASP)

- The Open Web Application Security Project (OWASP) is an organization focused on **improving** the security of software
- The mission of this organization is to make **software security visible**, so that individuals and organizations are able to make informed decisions



Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

OWASP TOP 10 Attacks-2017

- A1 – Injection
- A2 – Broken Authentication and Session Management
- A3 – Cross-site Scripting (XSS)
- A4 – Broken Access Control
- A5 – Security Misconfiguration
- A6 – Sensitive Data Exposure
- A7 – Insufficient Attack Protection
- A8 – Cross-site Request Forgery (CSRF)
- A9 – Using Components with Known Vulnerabilities
- A10 – Underprotected APIs

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

The Web Application Security Consortium (WASC)

- The Web Application Security Consortium (WASC) is an international group of **experts**, industry **practitioners**, and organizational **representatives** who produce open source and widely agreed upon best-practice security standards for the World Wide Web



Source: *http://www.webappsec.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

WASC Threat Classification

- **Threat Classification** is an effort to classify the **weaknesses**, and **attacks** that can lead to the compromise of a website, its data, or its users

Attacks		Weaknesses
Abuse of Functionality	Path Traversal	Application Misconfiguration
Brute Force	Predictable Resource Location	Directory Indexing
Buffer Overflow	Remote File Inclusion (RFI)	Improper File system Permissions
Content Spoofing	Routing Detour	Improper Input Handling
Credential/Session Prediction	Session Fixation	Improper Output Handling
Cross-Site Scripting	SOAP Array Abuse	Information Leakage
Cross-Site Request Forgery	SSI Injection	Insecure Indexing
Denial of Service	SQL Injection	Insufficient Anti-automation
Fingerprinting	URL Redirector Abuse	Insufficient Authentication
Format String	XPath Injection	Insufficient Authorization
HTTP Response Smuggling	XML Attribute Blowup	Insufficient Password Recovery
HTTP Response Splitting	XML External Entities	Insufficient Process Validation
HTTP Request Smuggling	XML Entity Expansion	Insufficient Session Expiration
Integer Overflows	XML Injection	Insufficient Transport Layer Protection
LDAP Injection	XQuery Injection	Server Misconfiguration
Mail Command Injection		
Null Byte Injection		
OS Commanding		

Source: *http://www.webappsec.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Software Security Framework: Software Assurance Maturity Model (SAMM)

- The Software Assurance Maturity Model (SAMM) is an open framework to help organizations formulate and implement a strategy for **software security** that is tailored for the specific risks faced by the organization

- **SAMM helps you:**
  1. Evaluate an organization's existing software security practices
  2. Build a balanced software security assurance program in well-defined iterations
  3. Demonstrate concrete improvements in the security assurance program
  4. Define and measure security-related activities throughout an organization

Source: *http://www.opensamm.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Software Security Framework: Software Assurance Maturity Model (SAMM) (Cont'd)

- The **maturity model** consists of four business functions:
  - **Governance**: Assess the management of application security in an organization
  - **Construction**: Assess the software creation process in an organization
  - **Verification**: Assess the software testing of the application
  - **Deployment**: Assess the deployment (Software release management) and production of the application
- Each function posses three security practices



Source: *http://www.opensamm.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Software Security Framework: Building Security In Maturity Model (BSIMM)

- The main objective of BSIMM is to enable the organization to **analyze** and **implement security features** required for the organization by evaluating most frequently implemented security features in other companies
- BSIMM is made up of a **software security framework** used to organize the **113 activities** used to assess initiatives
- The framework consists of **12 practices** organized into four domains

The Software Security Framework (SSF)			
Governance	**Intelligence**	**SSDL Touchpoints**	**Deployment**
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

Source: *https://www.bsimm.com*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

BSIMM vs OpenSAMM

BSIMM	OpenSAMM
Descriptive model	Prescriptive model
Has 12 security practices consisting of 112 activities	Has 12 security practices consisting of 72 activities
Based on things actually followed in an organization	Based on certain list of activities developed for software security
Has an active community that enables the organizations to understand the security features followed in other organizations	Do not have any active community activities

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Module Summary

- An application is said to be secured when its restricted resources and secrets are protected from malicious use
- Attackers are exploiting various application vulnerabilities to steal secrets or gain unauthorized access to their protected resources
- The network security perimeter has huge holes at the application layer
- Application are vulnerable to different types of attacks carried out at application level
- There are various reasons behind the existence of vulnerabilities in the applications
- Software application are designed and developed with functionality first in mind and security as a distant second or third
- A constant security vigilance is required at various phases of the development lifecycle to develop secure application
- Various Software Security Reference Standards, Models, Communities, and Frameworks can help in driving and maintaining application security

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

JAVA

C | ASE

Certified | Application Security Engineer



Module 02

Security Requirements Gathering

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

Module Objectives

1. Understand the Importance of Gathering Security Requirements
2. Understand Security Requirement Engineering (SRE) and its Phases
3. Describe Abuse Cases and Abuse Case Modeling
4. Describe Security Use Cases and Security Use Case Modeling
5. Describe Abuser and Security Stories
6. Explain Security Quality Requirements Engineering (SQUARE) Model
7. Explain Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) Model

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Importance of Gathering Security Requirements

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Security Requirements

- Software security requirements are **non functional** requirements, which need to be addressed to maintain the **confidentiality**, **integrity**, and **availability** of the application
- **Stakeholders** often overlook security requirement during the inception phase of software development
- This **negligence** may result in the application being vulnerable to different types of attacks or getting abused
- Gathering security requirements should be part of the **strategic application development process**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Gathering Security Requirements

1. **Eliciting** software security requirements takes different approach
2. It should be **enumerated** separate from the functional requirement so that they can be **reviewed** and **tested** separately
3. Mixing **security requirement** with **functional requirement** can make security requirement gathering process more **complicated** and **inaccurate**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Why We Need Different Approach for Security Requirements Gathering

1. Functional requirements are **positive requirements** specifying what the software should do
2. Security requirements are **negative requirements** specifying what the software should not do
3. It is **against** natural tendency of people that they are clear about what they want but quite find it difficult to understand what they don't want
4. A software needs to be viewed in a more **negative**, **critical**, and **destructive** way to reveal its non-intended use and its associated security requirements

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Key Benefits of Addressing Security at Requirement Phase

- Addressing security at requirement phase can save the economy **billions of dollars** as compared to addressing security at later phase of software development
- Security requirements give the developer an overview about **key security controls** required to build secure application
- It also specifies the **security mechanisms** that need to be implemented in order to comply with regulations, standards or requirements for the secure application development and attack protection
- Correctly understood security requirements can help in implementing security in **design**, **development**, and **testing** stages

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Stakeholders Involvement in Security Requirements Gathering

- **User, Customer or Business Sponsor**
- **Developers**
- **Requirement Engineer**
- **Security Expert**
- **Project Manager/Architects**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Characteristics of Good Security Requirement: SMART

Specific	Wording of the software requirement should be **clear** and **precise**. It should not be vague and all-encompassing
Measurable/Testable	There should be a **clear way** to test whether the specific requirement was met or not
Actionable	**Developer** should get clear understanding of what they exactly need to do to satisfy the requirements of the client
Realistic	It should be **implementable** in **real time** considering all the constraints
Timely	It should be of **high priority** when it is decided to implement

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Types of Security Requirements

- Security requirements are broadly classified into:

Functional Security Requirements

- It defines requirements specific to **business security**
- Example: Input validation, authentication, strong password, etc.

Security Drivers

- It defines requirements specific to **compliance** with **security policies**, **standards**, etc.
- Example: HIPPA, SOX, etc.

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Security Requirement Engineering (SRE)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Security Requirement Engineering (SRE)

- Security Requirement Engineering (SRE) empowers **security requirement gathering** in the application development process
- SRE is the **systematic approach** to identify, analyze, elicit, and specify security requirements for software system
- SRE plays a major role in integrated security **software development lifecycle process**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SRE Phases: Security Requirement Elicitation

- Requirement elicitation is the **initial activity** of any SRE approach

Typical actions in requirement elicitation:

1. **Analyze** the functional requirement of software system
2. Identify the relevant **stakeholders**
3. Gather security expectation required for various **functioning** of the system
4. Use different elicitation techniques such as brainstorming sessions, storyboarding, white boarding sessions, interviews, workshops, questionnaires, etc. to elicit security requirements

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

SRE Phases: Security Requirement Analysis

- Requirement analysis provides valid **blueprint** of security requirements elicited in earlier phase

Typical actions in requirement analysis:

1. Consider various aspects of the security requirements gathered in the **requirements elicitation phase**
2. Checks for any **conflicts** and **ambiguity** in the requirements
3. Check for its **completeness** and **clarity**
4. Check whether the requirement meets **stakeholder's expectation**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SRE Phases: Security Requirement Specification

- The requirement specification phase plays an **integral** part in the requirement engineering process as the later stages in the development of the application completely depends on the specifications provided in the document

Typical actions in requirement specification:

1. **Record** and **organize** the security requirements analyzed in requirement analysis phase in such a way that they can be clear, consistent, and traceable at any time
2. **Document** all the security requirements for future reference

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

SRE Phases: Security Requirement Management

- The security requirement management phase deals with maintaining the security requirements **gathered** and **documented**

Typical actions in requirement management:

1. **Update** of the security requirements
2. Provide support based on the selected **SRE approach**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Common Mistakes Made in Each Phase of SRE

Requirements Elicitation	Requirement engineer will not be able to **identified** all security requirements due to lack of knowledge of modern elicitation techniques or does not include all the relevant stakeholder during requirement phase
Requirements Analysis	The identified requirements are directly **specified** without any analysis or modeling
Requirements Specification	The identified requirements is not **SMART**
Requirements Management	The identified requirements is not **prioritized** or scheduled properly

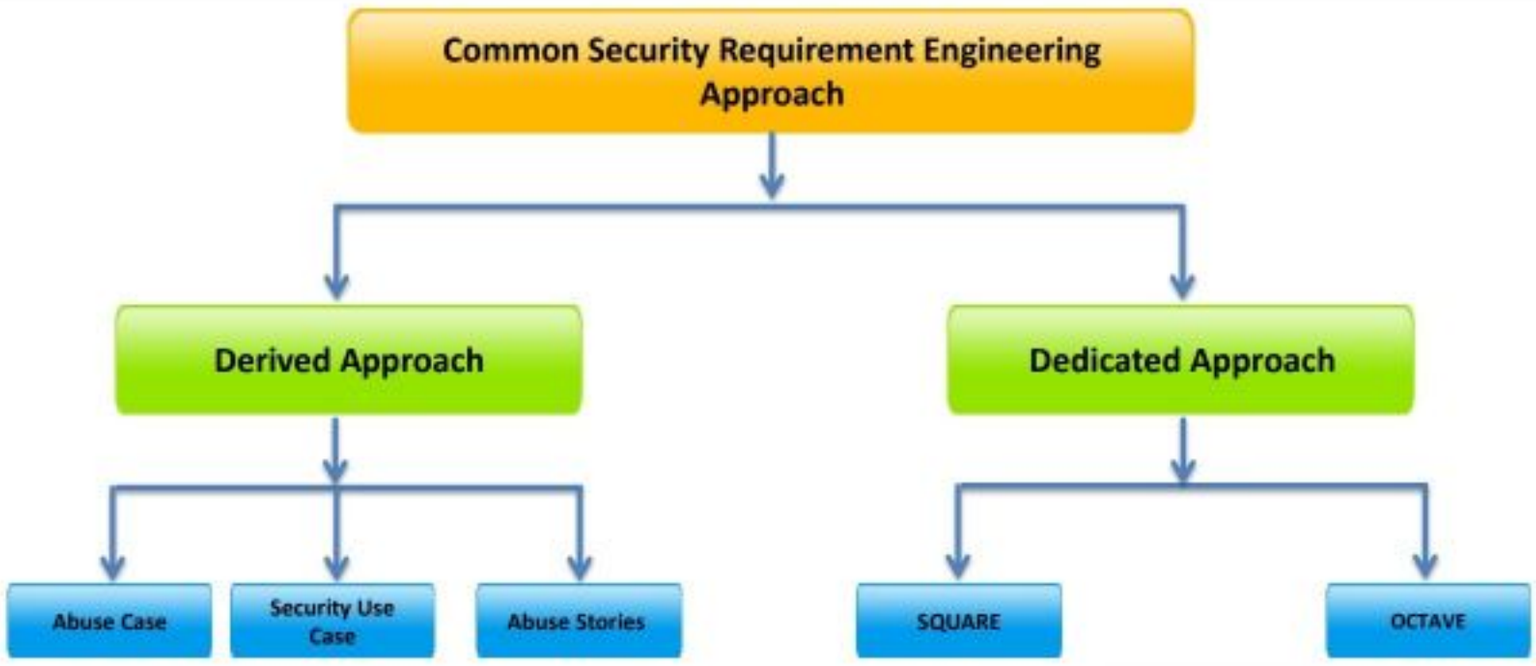
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Different Security Requirement Engineering Approaches/Model

- Typical SRE approaches are broadly classified into:
  - **Derived approach**: It is derived from traditional requirement gathering approach (use case modeling) to address security requirements in **software development process**
  - **Dedicated approach**: It is developed specifically to **address** security requirements



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Abuse Case and Security Use Case Modeling

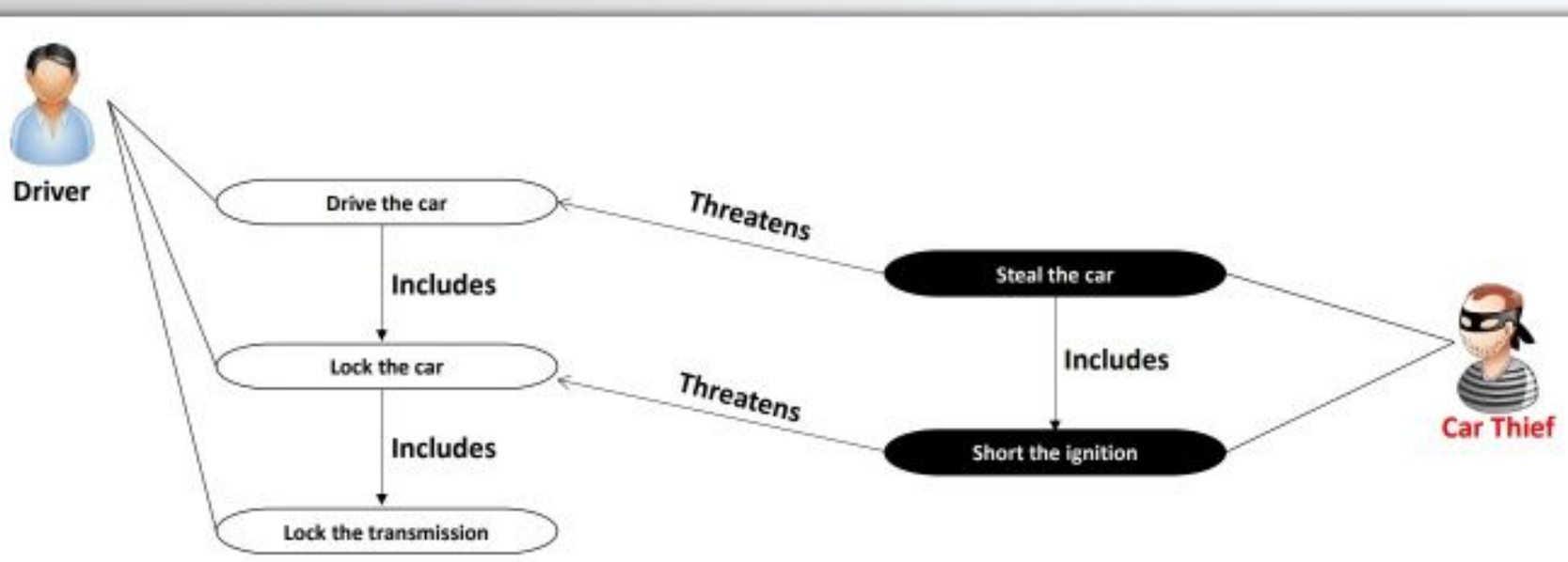
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Abuse Cases

- Abuse cases (or misuse case) derived from **use cases** of software system
- Abuse cases capture **abnormal behavior** of the system
- It depicts the actions taken by malicious users with the intent to **intrude** or **hack** or **compromise** the software functioning
- It helps **stakeholders** to differentiate between the appropriate and inappropriate use of software system
- Abuse cases lay the foundation for **threat modeling** in design phase



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Threatens Relationship

- The "threaten" relationship is used to describe **abuse case** scenario
- It describes the way the **attacker** can abuse the system
- It helps to **portray** the activity where an abuse case can threaten a use case with failure



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Abuse Case Modeling Steps

Step 1: Conduct **brainstorming** sessions or whiteboard sessions with relevant stakeholders and security experts

- Abuse case should be considered while developing use cases for functional requirements

Step 2: Analyze the use cases generated to capture **functional requirements**

- Start with High level use cases
- Continue to analyze use cases at granular level
- Understand the complete picture about what system does

Step 3: Think from **adversary's** perspective

- How system can be misused?
- How system can be disrupted?

Step 4: Identify the **malicious** actor

Step 5: Identify, specify, and define the **abuse** cases

Step 6: Build abuse case for every **use case**

Step 7: Check for **granularity** of detail

Step 8: Check **completeness** and **minimality**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Abuse Cases: Advantages and Disadvantages

Advantages

- It is easy for the managers to **learn** and **use** abuse cases
- Abuse cases contribute a lot during **requirement gathering phase**

Disadvantages

- Abuse cases can only be used with **object oriented software systems**
- Abuse case models do not specify any detail regarding the relation between use case and abuse case

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

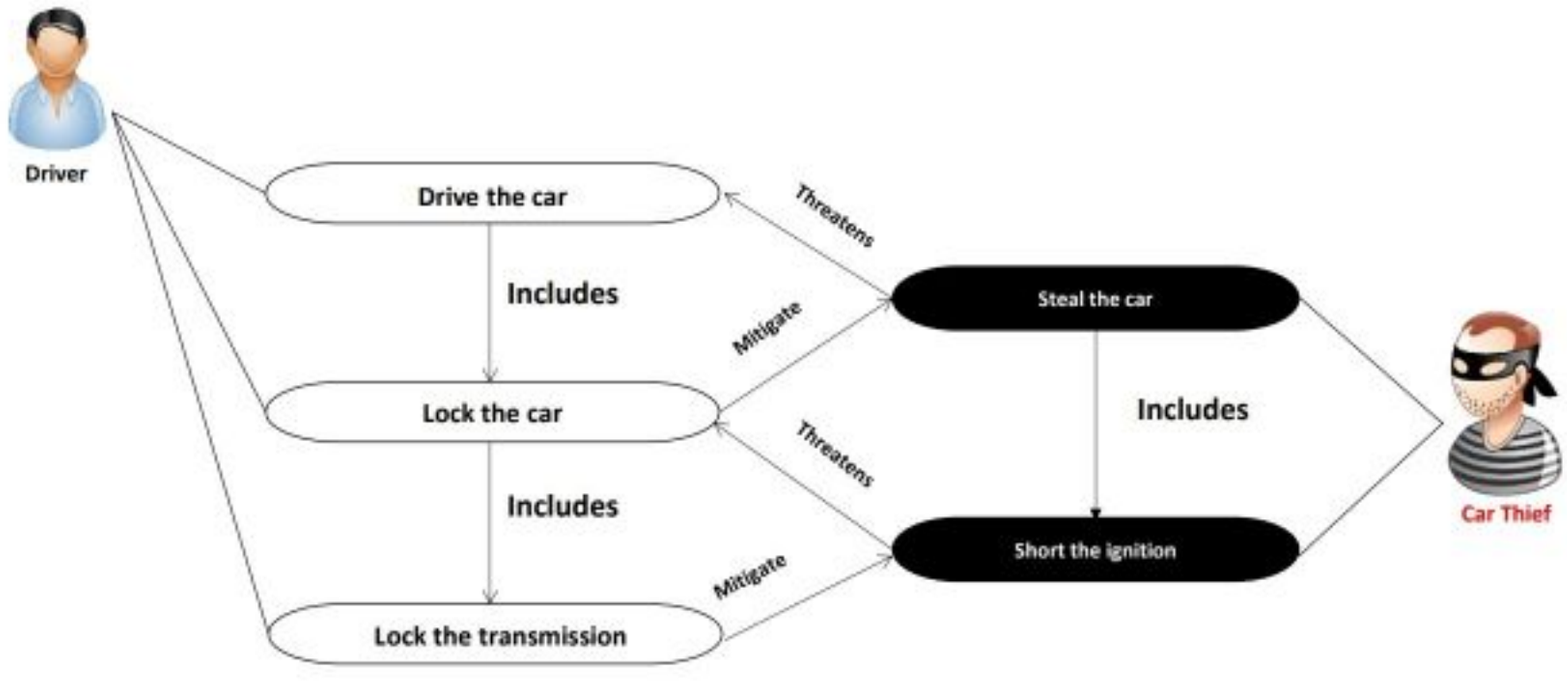
Abuse Case Template

Abuse Case ID:	
Name:	
Priority:	**Low or Medium or High**
Scope:	
Development Environment:	
Mis-actors:	
Access Right Levels:	**Low Level System Users or/and Medium Level System Users or/and High Level System Users or/and System Admin or/and Other Network User**
Point of Entry:	
Security Attributes Affected:	**Confidentiality or/and Integrity or/and Availability**
Description:	
Sophistication:	**Low or Medium or High**
Pre-conditions:	
Assumptions:	
Post-conditions:	
Potential Mis-actor Profiles:	
Stakeholders and Threats:	
Related Use Cases:	
Related Threats:	
Architectural Recommendation:	

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Security Use Cases

- Security use cases are derived from **abuse cases**
- Security use cases **capture** the security requirements needed to defeat each abuse case
- It **uncovers** software abuse cases and its response

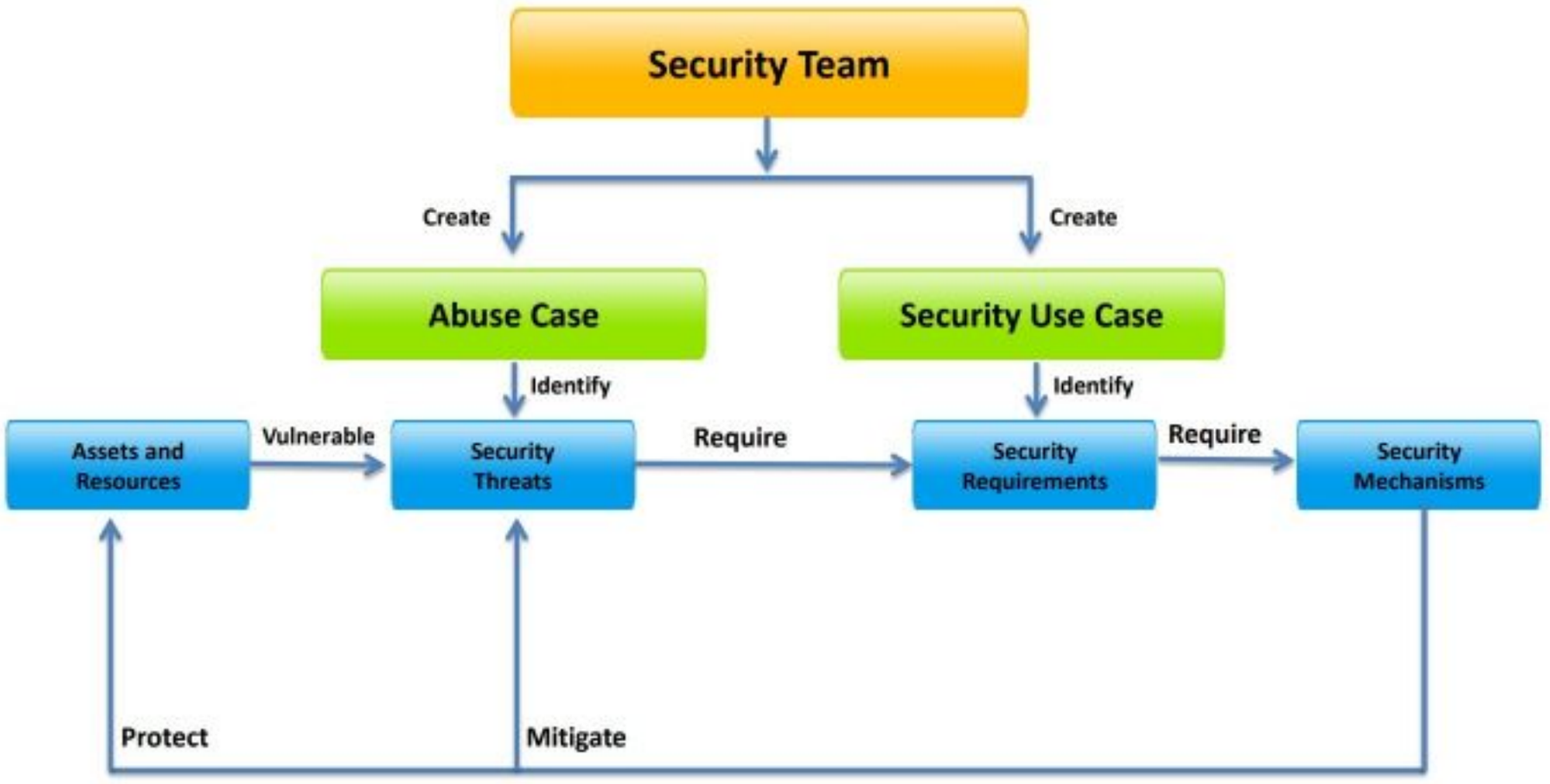


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Security Use Cases are Abuse Case Driven



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Modeling Steps for Security Use Cases

1. **Analyze** the possible abuse cases generated for system
2. Identify the security steps required to **mitigate** each abuse case at granular level
3. Include "**mitigation**" mappings between security use case and abuse case
4. Construct a **generalized** use case model consisting of use, abuse, and security use cases

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Mitigates Relationship

1. The mitigate relationship in a use case can **portray** the circumstances that can avoid the impact of any abuse case on any user or system
2. It provides **countermeasures** to uncover the abuse case



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Abuse Case vs Security Use Case

	Abuse Cases	Security Use Cases
Use	To specify security threats	To specify security requirements
Evaluation Criteria	Attacker succeeds	Application security succeeds
Created by	Security team	Security team
Consumed by	Security team	Requirements Team
Actors	Attacker	User
Driven By	Threat analysis	Abuse cases

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Security Use Case: Advantages and Disadvantages

Advantages

- **Large applications** mostly use security use cases
- Most of the security use cases depend on **scientific** and **precise** methods in order to define the security encounters in the system

Disadvantages

- Cannot be used for **small applications**
- Possibilities of **architectural** and **design** issues in the system

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Security Use Case Template

- Use **normal** use case template to describe security use cases

Typical security template consists of:

Use case name:	A unique name for the constructed use case that differentiates it from other use cases
Created By/-Author:	The person responsible for constructing the use case
Date Created/-Date:	The date on which the use case was constructed or created
Actor:	The person responsible for majority of the actions performed in the use case
Description/Summary:	This field provides details regarding the entire use case system in one or two lines
Preconditions:	This field specify those conditions that need to be satisfied by the system prior to the actual implementation of the use case
Post conditions:	This field describes the possible successful outcomes after the implementation of the use case
Basic/Normal course of events:	The field describes the most common path taken by the actors and the system to meet the objectives of the use case
Alternative courses/paths:	Here the less common path taken by the actors and system to meet the objectives of the use case is defined
Exceptions:	The exception field describes the path or the course taken by the actor and the system after the occurrence of an error
Assumptions:	The conditions specified are assumed to be true for the implementation of use case

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

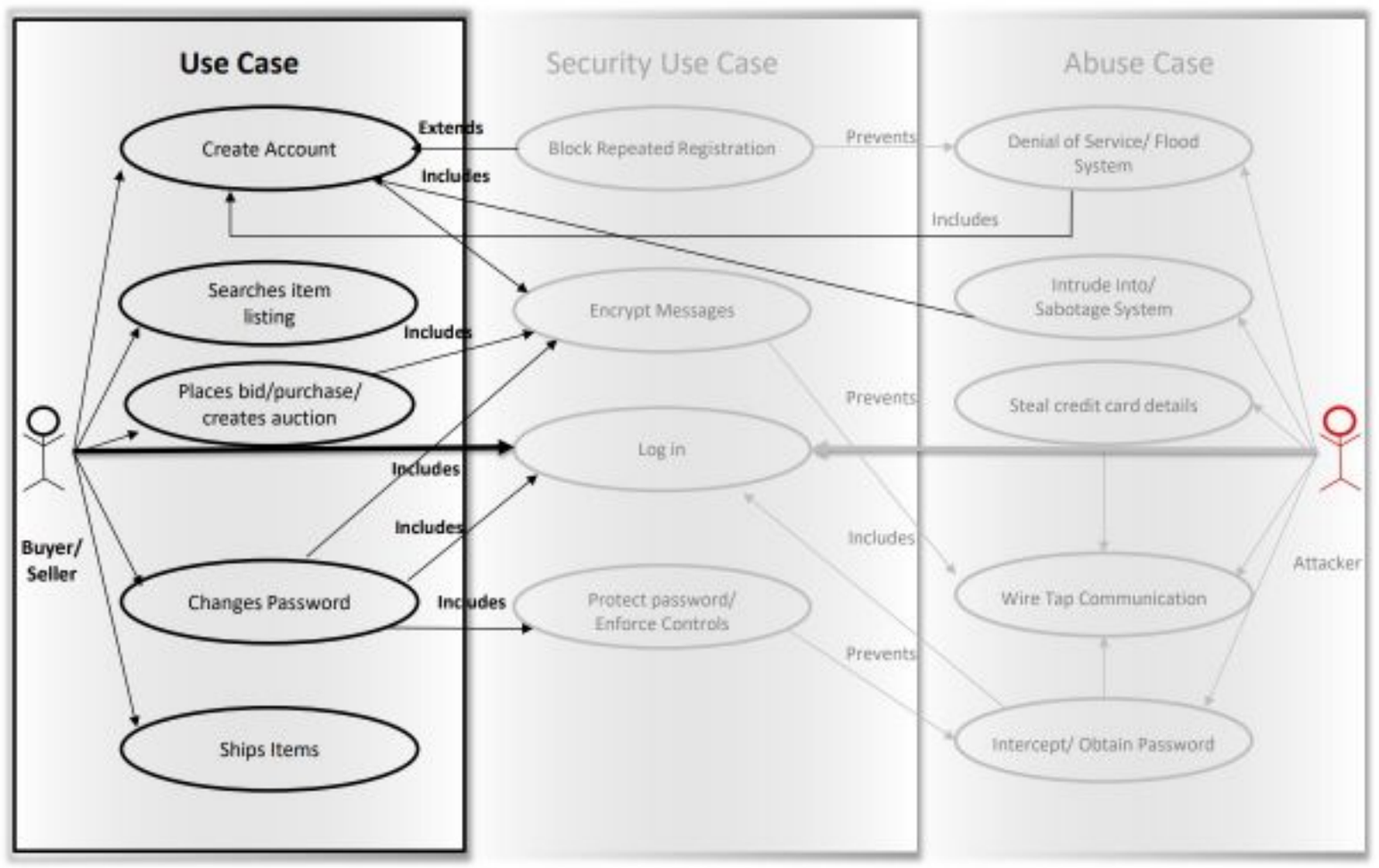
@ErrorCatchers

Security Use Case Guidelines

- Use cases should only specify the **vital requirements** needed for the development of the application
- Always ensure that the use cases do not include any **architectural mechanisms** like User ID, passwords, biometrics, etc.
- You may check the **structure** or **arrangement** of **interaction** of the user with the system and specify its order. In this way, it is possible to restrict any chances of evolution of the constraints in design
- It is advisable to create a base **security case** for various types of security requirements in the organization
- Always document the **threats** and **risks** with the help of security use cases
- The difference in **user** and **misuser interactions** must be specified in a transparent way in the security use cases
- The security use case should provide different scenarios for **externally-visible** actions and for those which are hidden
- Always ensure to clearly document both **preconditions** and **post conditions** in a security use case

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Example 1: Use Case for Online Bidding System

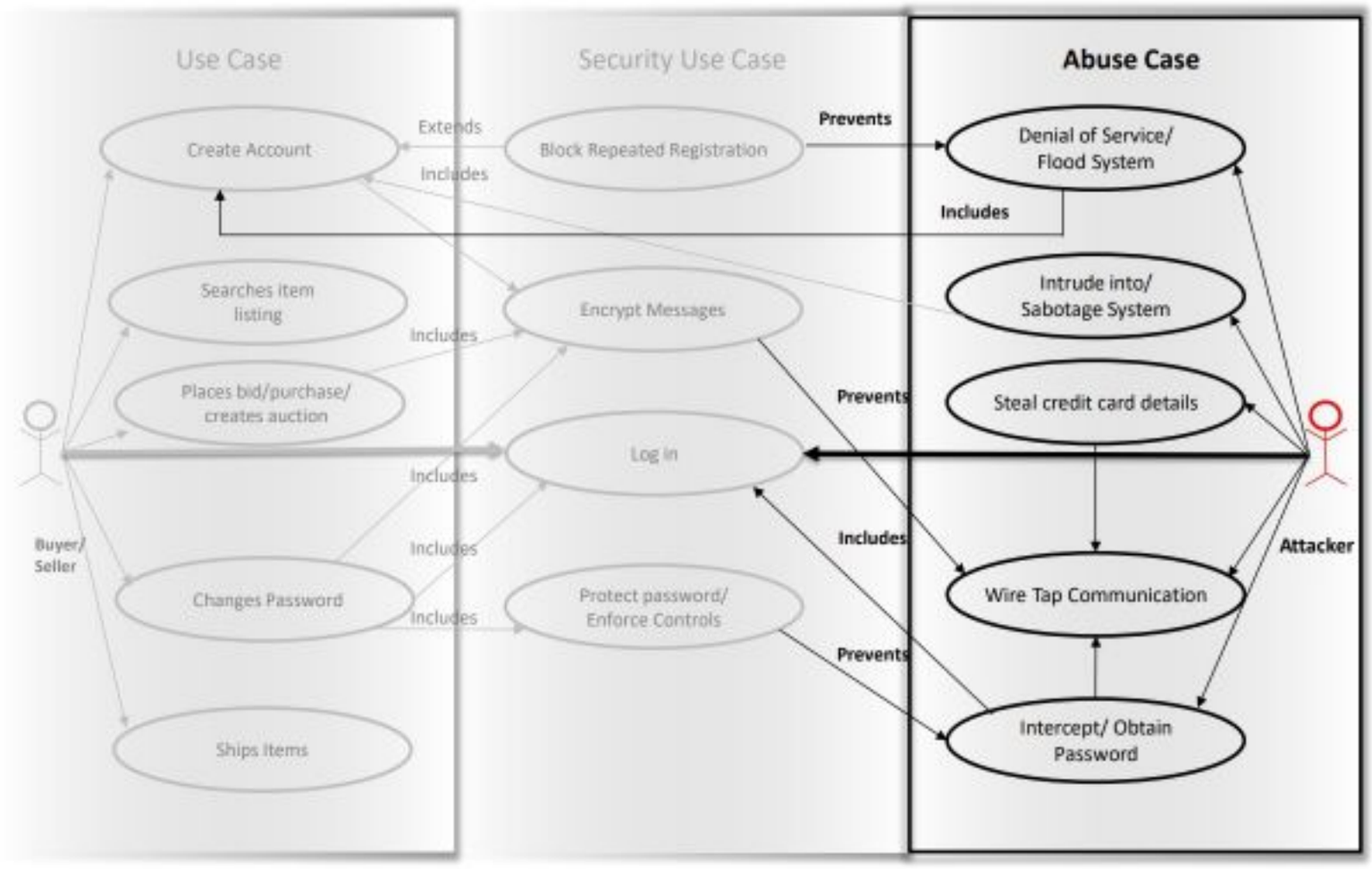


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

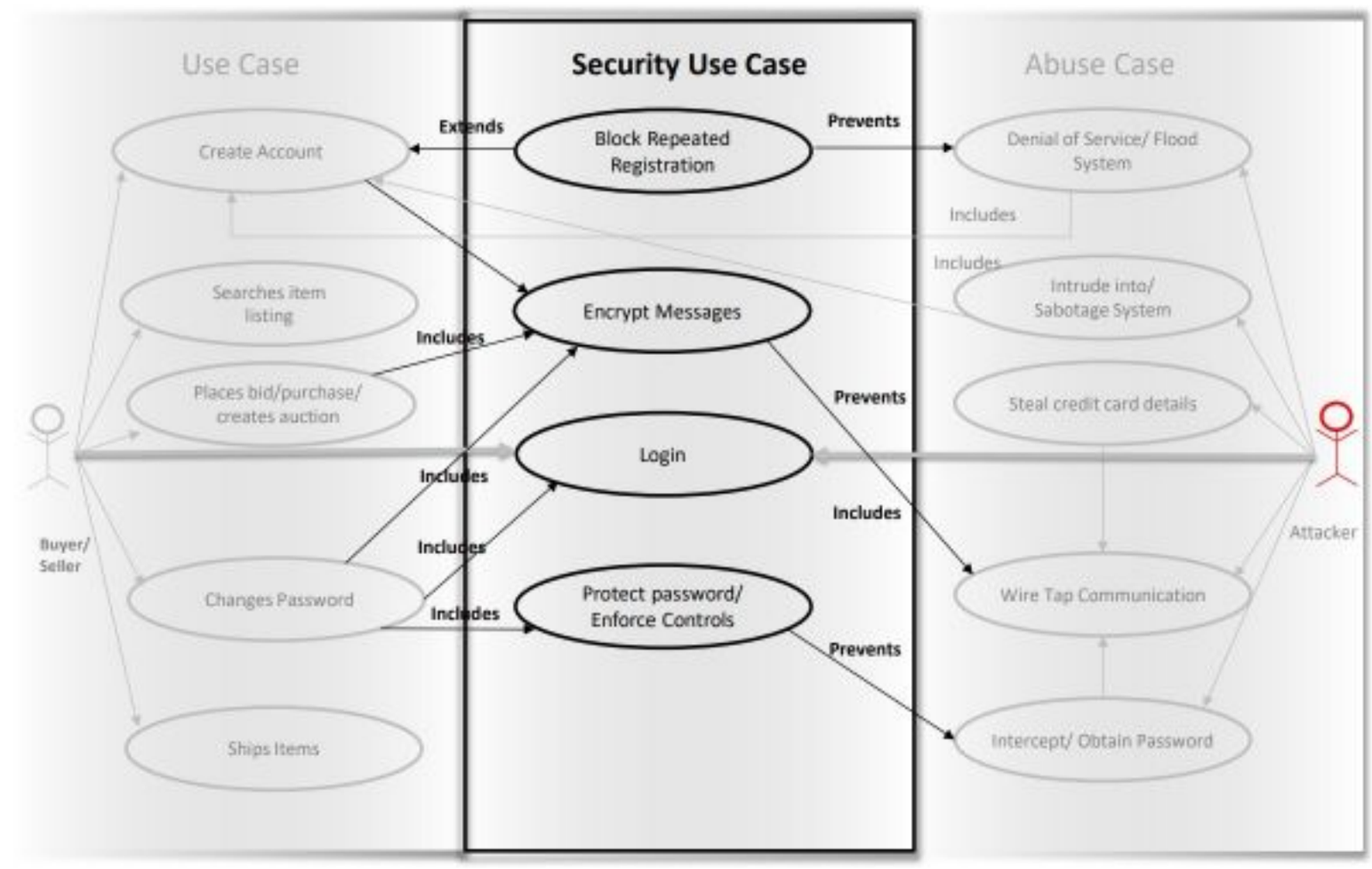
@ErrorCatchers

Example 1: Abuse Case for Online Bidding System



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Example 1: Security Use Case for Online Bidding System

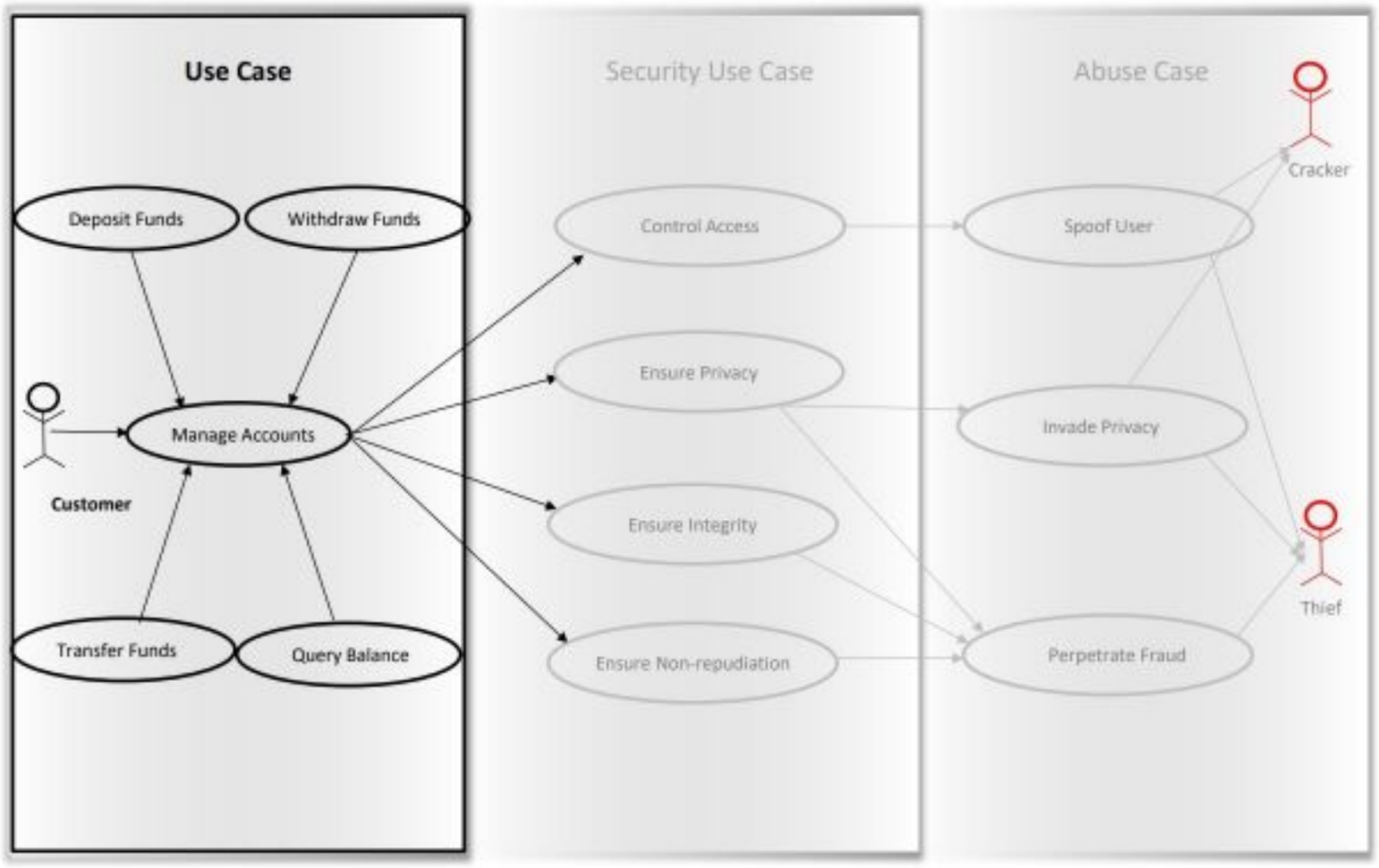


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

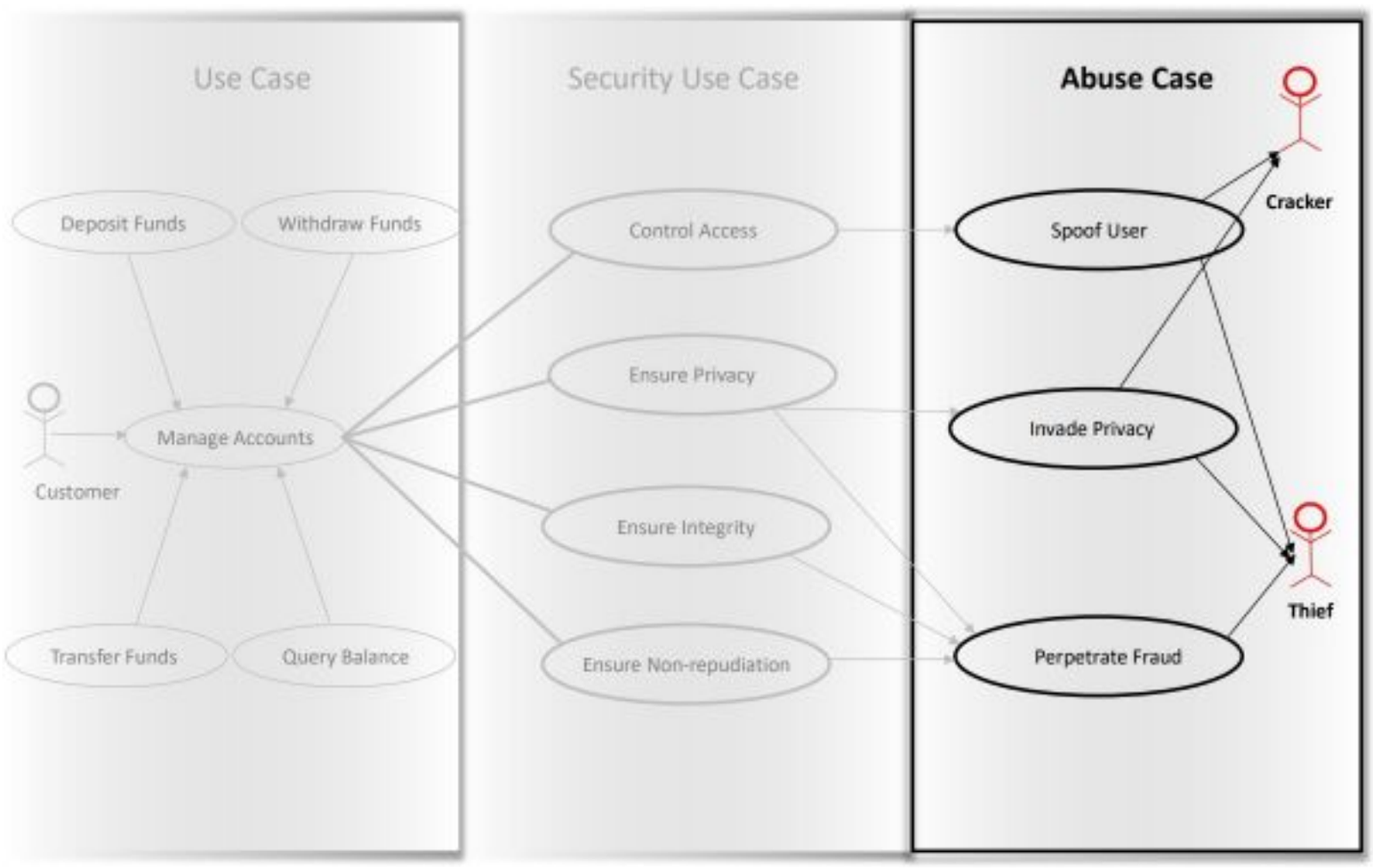
@ErrorCatchers

Example 2: Use Case for ATM System



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Example 2: Abuse Case for ATM System

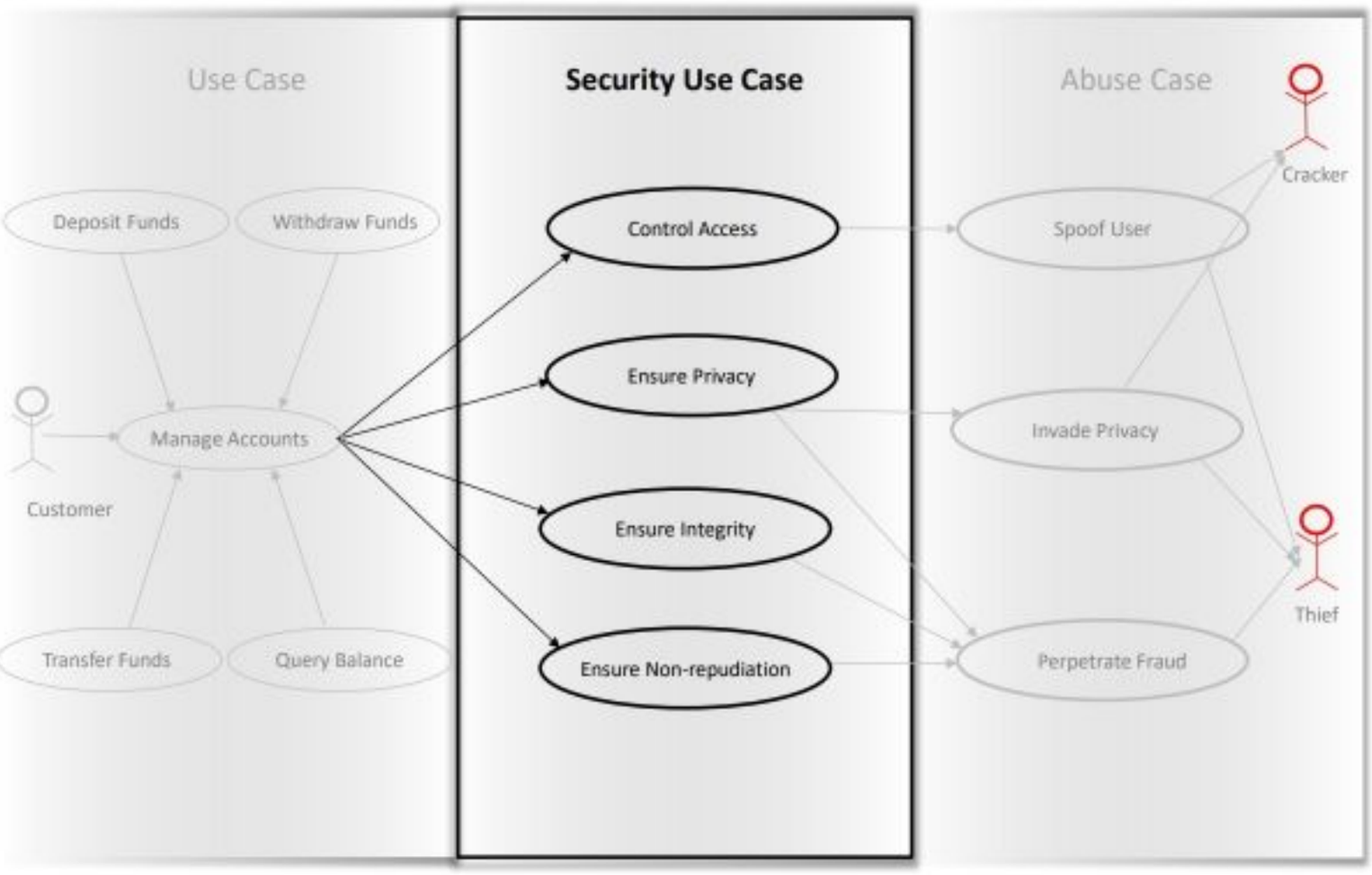


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

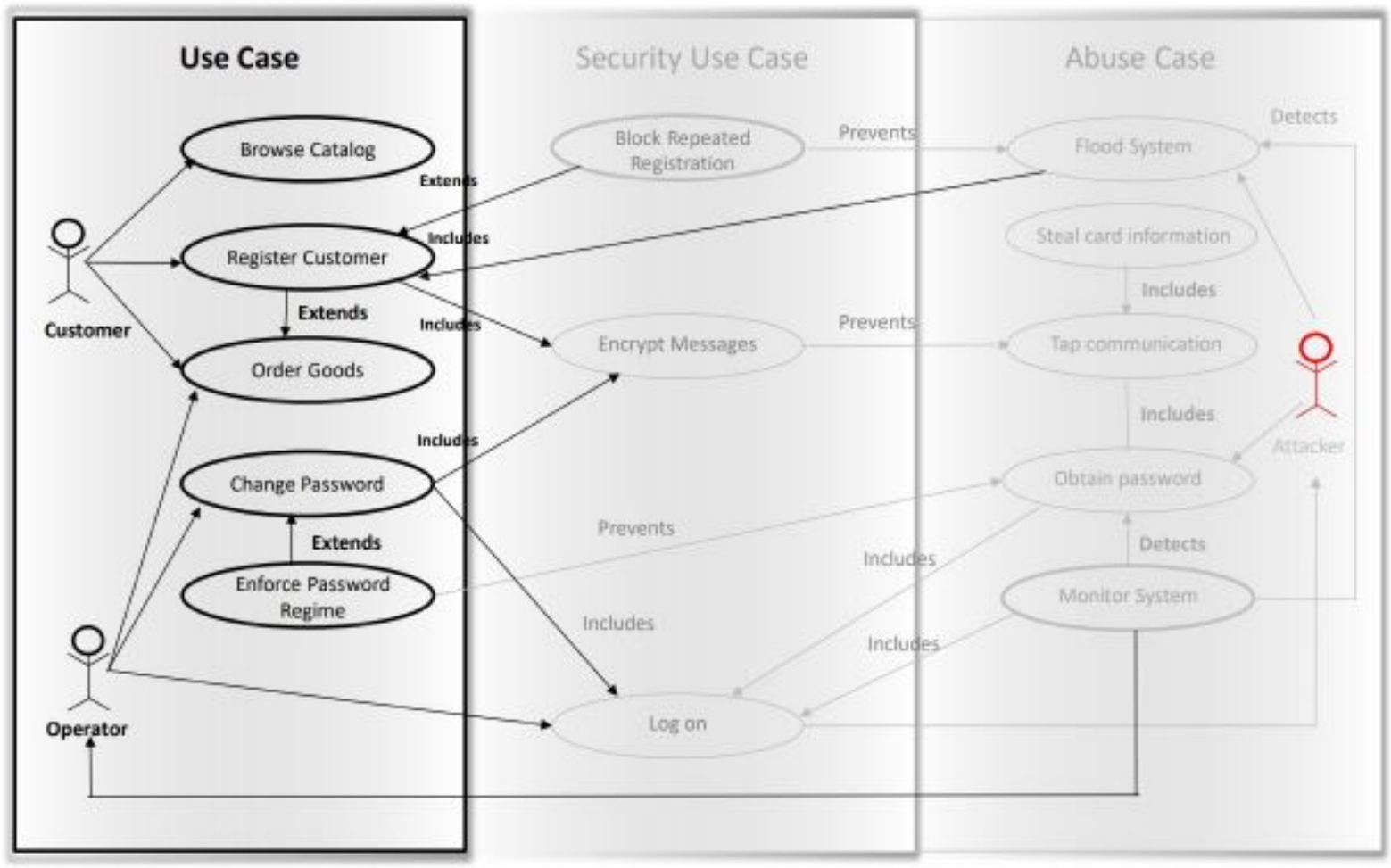
@ErrorCatchers

Example 2: Security Use Case for ATM System



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Example 3: Use Case for E-commerce System

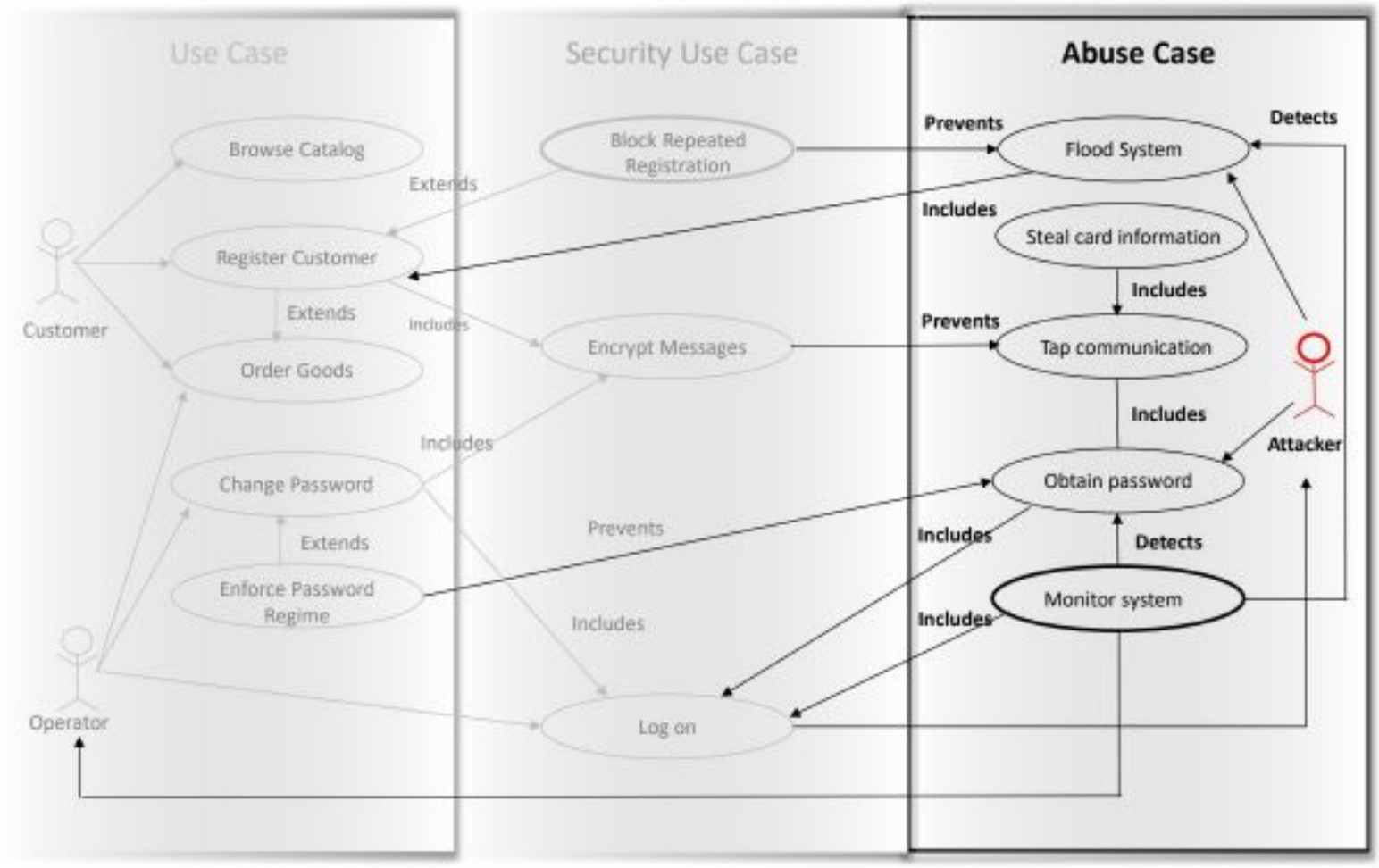


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

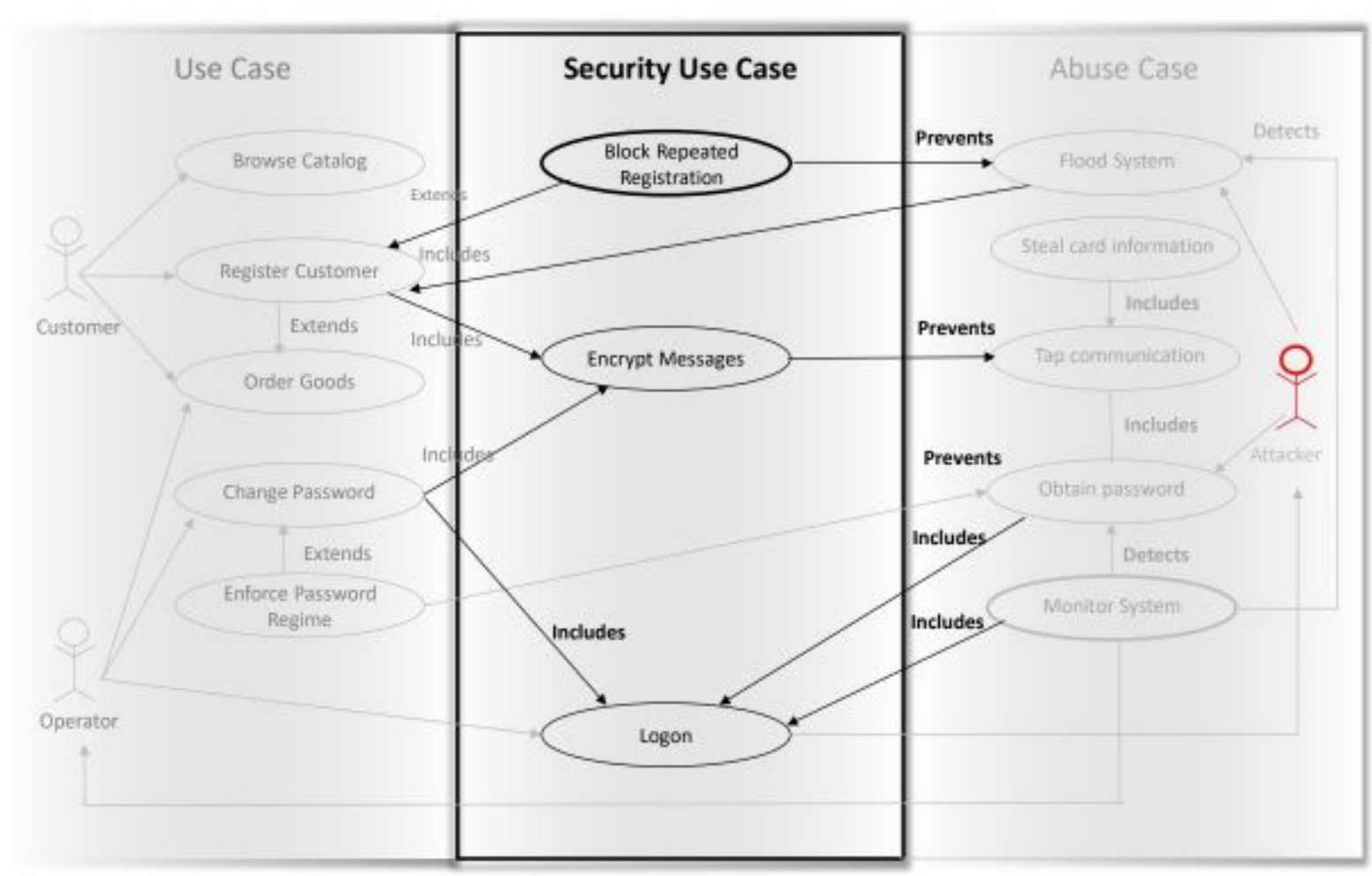
@ErrorCatchers

Example 3: Abuse Case for E-commerce System



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Example 3: Security Use Case for E-commerce System



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Effectiveness of Abuse and Security Case

Requirement Elicitation	Requirement Analysis
Degree of support provided for requirement elicitation: **High degree of support**	Types of analysis: **Internal and External analysis**
Method used for eliciting requirements: **Brainstorming including clients and developers**	Degree of unambiguity resolution of analysis: **Low level of detecting unambiguity and resolving it**
Degree of stakeholder identification provided: **Only moderate degree of stakeholder identification involved**	Degree of Completeness levels of analysis: **Only moderate degree of support available to ensure complete security requirements**
The level of involvement of the customer: **Ranges from moderate-high as misuse cases depend on customers**	Degree of clarity resolution level of analysis: **Low clarity level**
Elicitation of requirements other than security: **Non-functional requirements**	Degree of support level analysis for considering missed security requirements: **Moderate support level**
Dynamics of the elicitation process: **Iterative process**	Definite or Indefinite prevention of security requirement conflict: **Definite prevention**
Support for establishing system boundaries: **Moderate support**	

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Effectiveness of Abuse and Security Cases (Cont'd)

Requirement Specification	Requirement Management
Degree of usage of specification as a baseline for validation and verification of a system after its implementation: **Moderate usage**	Degree of difficulty level in making changes to security requirements: **Easy to make changes**
Degree of cost/time estimation support provided by specification: **Moderate support**	Degree of support level provided for accepting the evolving requirements: **Do not support**
Traceability level provided by the specification: **Low level**	Degree of support level provided for automating any step or process: **Low support**
Degree of consistency level provided by the specification: **Moderately consistent**	Degree of easiness in understandability provided by the approach for a novice user: **Easy to understand**
Degree of support for specifying non-functional requirements: **Moderate-high**	Degree of scalability level : **Low level**
Degree of clarity and understandability level provided by the specification: **Moderate level**	Degree of availability of information available: **Abundant information**
Degree of correctness or formality produced by the specification: **Informal specifications**	
Degree of exactness or rigor of the specification process: **Informal process**	
Degree of security level provided by the specification: **Moderate level of security**	

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Abuser and Security Stories

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Abuser and Security Stories

01 Abuser stories are the user stories written from the point of view of a **malicious adversary**

02 It is a **textual description** of the malicious interaction between malicious user and the software

03 Abuser stories are generally created through brainstorming sessions with relevant stakeholders

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Textual Description Template: Abuser Stories and Security Stories

Abuser Story Template: As (some kind of bad guy) I want to (do something bad)

Example: *"As a Malicious Hacker I want to steal credit card information so that I can make fraudulent charges"*

Security Story Template: As a (type of user) I want (something) so that (reason)

Example: *"As a(n) architect/developer, I want to verify that sensitive data is kept restricted to actors authorized to access it"*

Security Requirement Elicitation: *"Implement security for user information"*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Examples: Abuser Stories and Security Stories

Abuser Stories

- **Example #1**: "As a hacker I can send bad data in URLs, so that I can access data and functions for which I am not authorized"
- **Example #2**: "As a hacker I can send bad data in the content of requests, so that I can access data and functions for which I am not authorized"
- **Example #3**: "As a hacker I can send data in HTTP headers, so that I can access data and functions for which I am not authorized"
- **Example #4**: "As a hacker I can read and even modify all data that is input and output by your application"

Security Stories

- As a customer I want to have credentials protected
- As a customer I don't want my credentials to be public
- As an administrator I want to detect abuse of the system
- As an administrator I don't want the system to be misused
- As a controller I want data to be protected from unintended manipulation
- As a controller I don't want to have data manipulated by users unintendedly

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Effectiveness of Abuser and Security Stories

Requirement Elicitation	Requirement Analysis
Degree of support provided for requirement elicitation: **Moderate – High degree of support**	Types of analysis: **Internal analysis**
Method used for eliciting requirements: **Interviews**	Degree of unambiguity resolution of analysis: **Low level of detecting unambiguity and resolving it**
Degree of stakeholder identification provided: **High degree of stakeholder identification involved**	Degree of completeness levels of analysis: **Low degree of support available to ensure complete security requirements**
The level of involvement of the customer: **Ranges from moderate-high as customers validate and update abuser stories**	Degree of clarity resolution level of analysis: **Provides only Moderate level of clarity as customers constantly validates and updates abuser stories**
Elicitation of requirements other than security: **Do not support**	Degree of support level analysis for considering missed security requirements: **Do not support**
Dynamics of the elicitation process: **Iterative process**	Definite or indefinite prevention of security requirement conflict: **Do not support**
Support for establishing system boundaries: **Low support**	

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Effectiveness of Abuser and Security Stories (Cont'd)

Requirement Specification	Requirement Management
Degree of usage of specification as a baseline for validation and verification of a system after its implementation: **Low usage**	Degree of difficulty level in making changes to security requirements: **Easy to make changes**
Degree of cost/time estimation support provided by specification: **Moderate - High support**	Degree of support level provided for accepting the evolving requirements: **Do not support**
Traceability level provided by the specification: **High level**	Degree of support level provided for automating any step or process: **None**
Degree of consistency level provided by the specification: **Low Consistency level**	Degree of easiness in understandability provided by the approach for a novice user: **Moderate – Easy level**
Degree of support for specifying non-functional requirements: **Do not support**	Degree of scalability level : **Low level**
Degree of clarity and understandability level provided by the specification: **Moderate level**	Degree of availability of information available: **Less information**
Degree of correctness or formality produced by the specification: **Informal specifications**	
Degree of exactness or rigor of the specification process: **Informal process**	
Degree of security level provided by the specification: **Low level of security**	

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Abuser Stories: Advantages and Disadvantages

Advantages

- Abuser stories identify the **characteristics** of a system when under threat or failure
- It helps in documenting all the **flaws** and **gaps** in a system that needs to be rectified

Disadvantages

- Abuser stories are best applicable only in **agile process**
- It is not possible to completely secure the system using abuser stories approach

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Security Quality Requirements Engineering (SQUARE)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Security Quality Requirements Engineering (SQUARE)

- System Quality Requirements Engineering (SQUARE) is a process model that was developed at **Carnegie Mellon University**

- Security Quality Requirements Engineering (SQUARE) provides a means for **eliciting**, **categorizing**, and **prioritizing** security requirements for information technology systems and applications

- The focus of this methodology is to build security concepts into the early stages of the development life cycle

Source: *https://www.us-cert.gov*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQUARE Effectiveness

Requirement Elicitation	Requirement Analysis
Degree of support provided for requirement elicitation: **Moderate – Low degree of support**	Types of analysis: **Internal and External analysis**
Method used for eliciting requirements: **Any technique like Interviews and Surveys**	Degree of unambiguity resolution of analysis: **High level of detecting unambiguity and resolving it**
Degree of stakeholder identification provided: **Moderate - High degree of stakeholder identification involved**	Degree of completeness levels of analysis: **Moderate level of support available to ensure complete security requirements**
The level of involvement of the customer: **High involvement of customers**	Degree of clarity resolution level of analysis: **Provides only High level of clarity**
Elicitation of requirements other than security: **Functional and Non-functional**	Degree of support level analysis for considering missed security requirements: **Low level of support**
Dynamics of the elicitation process: **Iterative process**	Definite or indefinite prevention of security requirement conflict: **Definite prevention**
Support for establishing system boundaries: **Low support**	

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

SQUARE Effectiveness (Cont'd)

Requirement Specification	Requirement Management
Degree of usage of specification as a baseline for validation and verification of a system after its implementation: **Moderate – High usage**	Degree of difficulty level in making changes to security requirements: **Moderate level**
Degree of cost/time estimation support provided by specification: **High support**	Degree of support level provided for accepting the evolving requirements: **Moderate - High**
Traceability level provided by the specification: **Low level**	Degree of support level provided for automating any step or process: **Low**
Degree of consistency level provided by the specification: **Moderate – High Consistency level**	Degree of easiness in understandability provided by the approach for a novice user: **Moderate difficulty**
Degree of support for specifying non-functional requirements: **Low support**	Degree of scalability level : **High level**
Degree of clarity and understandability level provided by the specification: **Moderate level**	Degree of availability of information available: **Abundant information**
Degree of correctness or formality produced by the specification: **Semi_ Formal specifications**	
Degree of exactness or rigor of the specification process: **Semi – Formal process**	
Degree of security level provided by the specification: **Moderate – High level of security**	

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQUARE Process

Number	Step	Input	Techniques	Participants	Output
1	Agree on definitions	Candidate definitions from IEEE and other standards	Structured interviews, focus group	Stakeholders, requirements engineer	Agreed to definitions
2	Identify assets and security goals	Definitions, Candidate goals, Business drivers, Policies and procedures, Examples	Facilitated work session, surveys, interviews	Stakeholders, requirement engineer	Assets and goals
3	Develop artifacts to support security requirements definition	Potential artifacts (e.g.: scenarios, misuse cases, templates, forms)	Work session	Requirements engineer	Needed artifacts: scenarios, misuse cases, models, templates, forms
4	Perform risk assessment	Misuse cases, scenarios, goals	Risk assessment method, analysis of anticipated risk against organizational risk tolerance, including risk analysis	Requirements engineer, stakeholder, risk expert	Risk assessment results

Source: *https://www.us-cert.gov*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

SQUARE Process (Cont'd)

Number	Step	Input	Techniques	Participants	Output
5	Select elicitation techniques	Goals, definitions, candidate techniques, expertise of stakeholders, organizational style, culture, level of security needed, cost/benefit analysis, etc.	Work session	Requirements engineer	Selected elicitation techniques
6	Elicit security requirements	Artifacts, risk assessment results, selected techniques	Joint application development, surveys, interviews, model-based analysis, checklists, lists of reusable requirements types, document reviews	Stakeholders facilitated by requirement engineer	Initial out at security requirements
7	Categorize requirements as per level (system software, etc.) and whether they are requirements or other kinds of constraints	Initial requirements, architecture	Work session using a standard set of categories	Requirements engineer, other specialists as needed	Categorized requirements
8	Prioritize requirements	Categorized requirements and risk assessment results	Prioritization methods such as analytical hierarchy process (AHP) Triage, Win-Win	Stakeholders facilitated by requirement engineer	Prioritized requirements
9	Inspect requirements	Prioritized requirements, candidate formal inspection technique	Inspection method such as Fagan, Per-reviews	Inspection team	Initial selected requirements, documentation of decision-making process and rationale

Source: *https://www.us-cert.gov*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQUARE: Advantages and Disadvantages

Advantages

- Enables **easy understanding** of the system and at the same time is easy to learn
- **Inexpensive** to implement

Disadvantages

- Only useful for **large projects**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE)

- OCTAVE is an **information risk evaluation** method that enables organizations to determine the risk factors affecting the confidentiality, integrity, and availability of assets
- OCTAVE provides a **structured approach** for identifying, prioritizing, and managing the security risks in an organization
- It also enables the organization to make a better understanding of the security requirements needed for **maintaining security** of the assets

The OCTAVE approach consists of three phases:

1. **Phase 1**: Identifying the asset based threat
2. **Phase 2**: Evaluating and identifying the vulnerabilities
3. **Phase 3:** Drafting security strategies and plans

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

OCTAVE Effectiveness

Requirement Elicitation	Requirement Analysis
Degree of support provided for requirement elicitation: Moderate – **High degree of support**	Types of analysis: **High Internal analysis**
Method used for eliciting requirements: **Workshops**	Degree of unambiguity resolution of analysis: **None as only low level of support is provided to clear the unambiguity in the requirements gathered**
Degree of stakeholder identification provided: **High degree of stakeholder identification involved**	Degree of completeness levels of analysis: **Moderate level of support available**
The level of involvement of the customer: **High involvement of customers as they mostly participate in eliciting security requirements as well as gathering information required for the effectiveness of OCTAVE**	Degree of clarity resolution level of analysis: **Provides only Moderate – High level of clarity**
Elicitation of requirements other than security: **Non-functional requirements**	Degree of support level analysis for considering missed security requirements: **Low level of support**
Dynamics of the elicitation process: **Iterative process**	Definite or indefinite prevention of security requirement conflict: **Low prevention as it does not explain the steps for the developers to clear the conflict**
Support for establishing system boundaries: **Moderate support**	

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

OCTAVE Effectiveness (Cont'd)

Requirement Specification	Requirement Management
Degree of usage of specification as a baseline for validation and verification of a system after its implementation: **Moderate usage**	Degree of difficulty level in making changes to security requirements: **High level**
Degree of cost/time estimation support provided by specification: **High support**	Degree of support level provided for accepting the evolving requirements: **Do not support**
Traceability level provided by the specification: **Moderate level**	Degree of support level provided for automating any step or process: **Do not support**
Degree of consistency level provided by the specification: **Low Consistency level**	Degree of easiness in understandability provided by the approach for a novice user: **Difficult**
Degree of support for specifying non-functional requirements: **Do not support**	Degree of scalability level: **High level**
Degree of clarity and understandability level provided by the specification: **Moderate - High level**	Degree of availability of information available: **Abundant**
Degree of correctness or formality produced by the specification: **Semi - Formal specifications**	
Degree of exactness or rigor of the specification process: **Formal process**	
Degree of security level provided by the specification: **Moderate level of security**	

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

OCTAVE Steps

- **The steps involved in OCTAVE security requirement process are:**

1. Identify the **important** and **valuable** assets
2. Determine the **security objective** for those assets
3. Identify and analyze the **threats** and **risks** involved in the identified critical assets
4. Determine security **requirements** for the identified threats and risks

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

OCTAVE: Advantages and Disadvantages

Advantages

- It enables **small** and **large** organizations to apply OCTAVE approach during security elicitation process

Disadvantages

- It is difficult to **understand** and **implement** the concepts of OCTAVE approach
- The approach does not provide a detailed explanation of the **quantitative analysis** of the security risks in the organization

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Module Summary

- Security requirements is negative requirements specifying what software should not do
- Gathering security requirements should be part of the strategic application development process
- Eliciting software security requirements takes different approach
- Security requirements should be enumerated separately from functional requirement so that they can be reviewed and tested separately
- Security Requirement Engineering (SRE) empowers security requirement gathering in the application development process

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

JAVA

C | ASE

Certified | Application Security Engineer

Module 03

Secure Application Design and Architecture

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

Module Objectives

1. Understand the Importance of Secure Application Design
2. Discuss Various Secure Design Principles
3. Understand Threat Modeling
4. Describe Threat Modeling Process
5. Discuss STRIDE and DREAD Model
6. Describe Secure Application Architecture Design

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Most of the web applications are vulnerable due to insecure design of application at design phase.

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

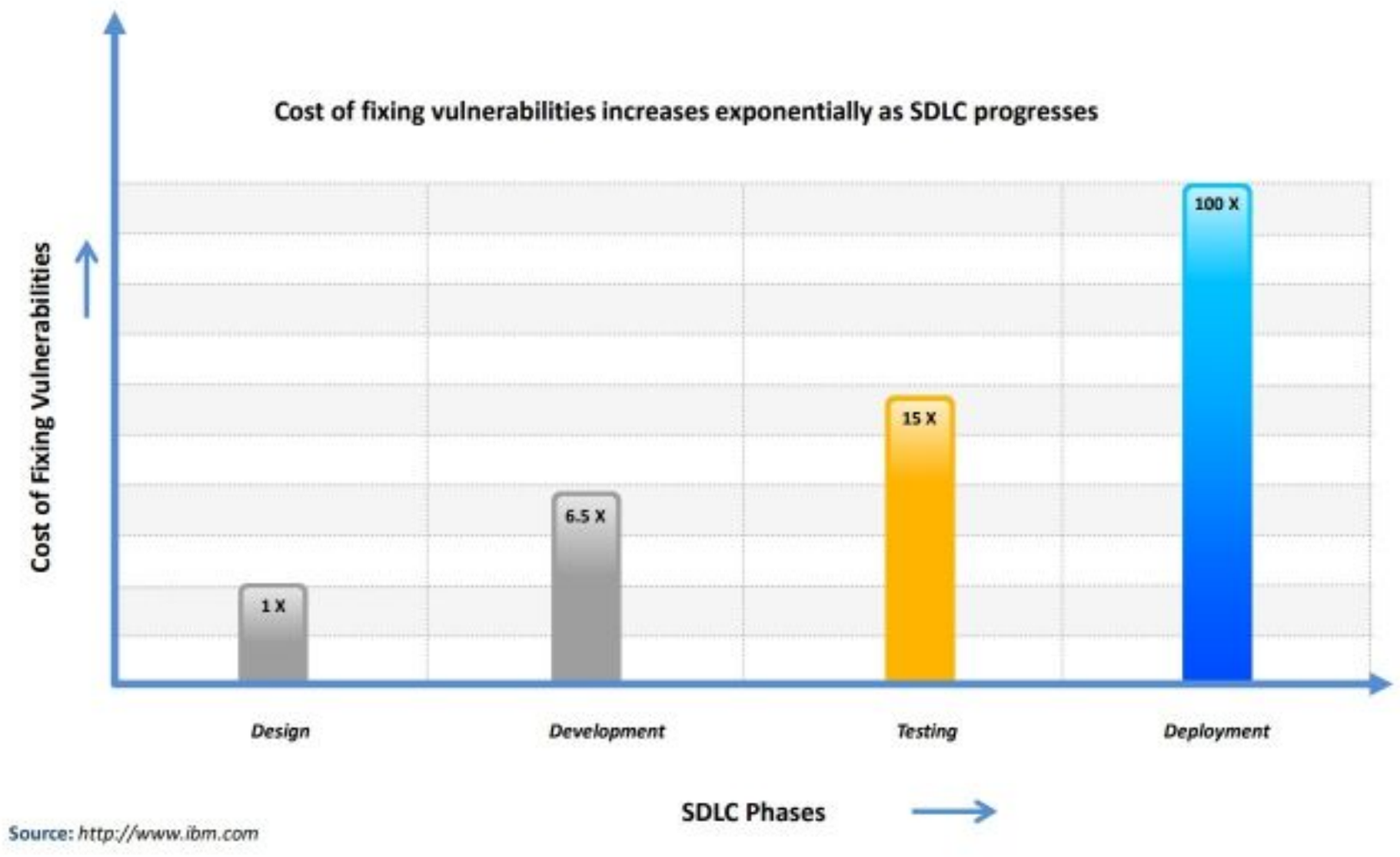
@ErrorCatchers

"Secure programming cannot make up for a poor design."

-- PHP Security Consortium

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Relative Cost of Fixing Vulnerabilities at Different Phases of SDLC



Source: http://www.ibm.com

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Application Design and Architecture

- A security negligence at **design** and **architecture** phase may lead to vulnerabilities that are difficult to detect and expensive to fix in production
- Security vigilance at design phase enables detecting potential **security flaws** early in the software development lifecycle
- Secure design of an application is based on **security requirements** identified in the previous phase of the SDLC
- Secure design is a **challenging process** as designing required security controls may obstruct the business functionality requirements

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Goal of Secure Design Process

I. Identifying the threats in sufficient details for **developers** to understand and code accordingly to mitigate the risk associated with the threats

II. Designing a **architecture** in such a way that it mitigates as many threats as possible

III. Enforcing **secure design principles** that force developers to consider security while coding

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Design Actions

Security Requirement Specifications

Design the application according to security specifications gathered at **requirement phase**

(Covered in Module 02: Security Requirements Gathering)

Secure Design Principles

Define the secure coding standards to be implemented in **development phase**

Threat Modeling

Perform threat modeling to know your **threats**

Secure Application Architecture

Design secure **application architecture**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secure Design Principles

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Define Secure Design Principles

- Secure design principles are the state of **practices or guidelines** that should be enforced on the developers to follow during development phase
- It helps in deriving **secure architectural decisions**
- It helps to eliminate design and architecture **flaws** and mitigate common security vulnerabilities within the application

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secure Design Principles

- **List of secure design principles to prevent common security vulnerabilities:**
  - Security through obscurity
  - Secure the weakest link
  - Use least privilege principle
  - Secure by default
  - Fail securely
  - Apply defense in depth
  - Do not trust user input
  - Reduce attack surface
  - Enable auditing and logging
  - Keep security simple
  - Separation of duties
  - Fix security issues correctly
  - Apply security in design phase
  - Protect sensitive data
  - Exception handling
  - Secure memory management
  - Protect memory or storage secrets
  - Fundamentals of control granularity
  - Fault tolerance
  - Fault detection
  - Fault removal
  - Fault avoidance
  - Loose coupling
  - High cohesion
  - Change management and version control

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Design Principles (Cont'd)

Security through Obscurity

- Security Through Obscurity (STO) relies on **preventing access** to certain users to protect internal data
- STO systems may have theoretical or actual security vulnerabilities, but designers believe that **flaws are unknown** and attackers are unlikely to find them
- Its usefulness has declined with the rise of **open systems**, networking, greater understanding of **programming techniques**, and increased capabilities of home users

Secure the Weakest Link

- Attackers target a system that is easy to **penetrate**
- For example, to gain access to the **encrypted** data on the network, attackers will not intercept the data and crack encryption; instead they will go after the **end points** of communication to find a **flaw** that discloses the data
- Identify and strengthen the **areas at risk** until levels of risk are satisfactory

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secure Design Principles (Cont'd)

Use Least Privilege Principle

- Applications with maximum **system privileges** are vulnerable to the **attacks**
- For example: Many web applications use **database admin account** though not required to connect to the backend database, enhancing the impact of SQL injection exploits
- Protects application from malicious attacks by:
  - Determining and assigning rights only to those who require **privileges** to complete the specific task
  - Avoiding applications that get **installed** and **run** by **default**
  - Writing applications that can be used by users having **non-administrative privileges**

Secure by Default

- The software solution or application provided to the users should be **security enabled** by default. If permitted, it is up to the user to reduce the security
- For example: By default the security feature **password aging** and **complexity** should be enabled

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Design Principles (Cont'd)

Fail Securely

- The developer should not give application secrets by default **error messages**
- Application that discloses **confidential information** on failure assists attackers in creating an **attack**
- When an application fails, determine what may occur and ensure that it does not threaten the application
- Provide logical and useful **error messages** to the users and store the details in the **log file**

Apply Defense in Depth

- The architects and developers should consider all the **levels of the software** to impose security while developing software
- Implement security mechanisms at **different layers** that include network layer, kernel layer, physical layer, and the file system layer

Do Not Trust User Input

- Protect the application from all **malicious inputs** coming from the user input to the application
- Consider all inputs as a malicious input and apply **security measures** to restrict them

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secure Design Principles (Cont'd)

Reduce Attack Surface

- Application attack surface area is to be minimized by **reducing the number of entry points** into the application
- Remove or turn off the features, protocols, and functionality which are **not in use** to minimize number of vulnerabilities and the overall risk
- For example: If a vulnerability exists in a way an XML is parsed, denying XML from unknown users minimizes that security vulnerability

Enable Auditing and Logging

- Auditing and logging states how the **security related events** are recorded by an application
- Auditing enables **identification of attacks or intruders in progress**, whereas logging aids in identifying how an attack is performed
- Perform auditing and logging to gather information about **attacks**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Design Principles (Cont'd)

Keep Security Simple

- If the design is complicated, it is hard to understand and **errors** are likely to occur in implementation, configuration, and use
- On the other hand, if the **complexity** of security mechanisms increases, the effort required to reach the appropriate level of **software assurance** also increases
- Avoid complex **architectures** and opt for **simpler approaches** that are fast and simple

Separation of Duties

- Separation of duties is the **key control of fraud**
- When assigning privileges, system roles are to be considered. In general, system administrators are also the users as some super user privileges are required to make the system run
- For example: **System administrator** can set the password policy, turn off or on the system, etc. but should not be able to log in as a **super-privileged** user

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secure Design Principles (Cont'd)

Fix Security Issues Correctly

- When a security issue is identified, fix it, considering it as the actual problem, and then go through the **security process** as you do for the new code, ensuring that the fix does not introduce new errors
- **For example**: User capable of viewing **another user's account balance** by simply adjusting cookie. In this context once the security issue is fixed it has to be tested on all the applications as cookie handling code is shared among all applications

Apply Security in Design Phase

- Before starting the application development process, always consider **security issues** that can help prevent many **security vulnerabilities**
- Considering security issues helps you understand the **coding weaknesses** and **vulnerabilities** from the most obvious exploits

Protect Sensitive Data

- Do not **hard code** the sensitive data such as passwords in the program
- Use **data encryption mechanism** to transmit data over the network

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Design Principles (Cont'd)

Exception Handling

- Events that disrupt the **coding process** are called exceptions
- Exception handling occurs when error conditions **interrupt** the normal flow of a program's execution
- Programmers have difficulty in designing for exception handling, as continuous checking for **error conditions** is necessary
- Proper use of exception handling helps to ensure proper **error handling**

Secure Memory Management

- Check memory bounds on the length of input variables, arrays, and arguments to prevent **buffer overflow attacks**
- Apply coding standards for simplicity, which help in **implementing security** in the **program** and keep things simple

Protect Memory or Storage Secrets

- **Encrypt secrets** to protect memory storage from ending up in crash dump file
- Use a perfect **cryptographic method** to perform encrypting secrets process
- Scrub secrets in memory storage before **deletion**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secure Design Principles (Cont'd)

Fundamentals of Control Granularity

- Applications managing sensitive data should possess **privacy protection** that can be achieved through flow control
- **Flow control** prevents data from non-secure information flows
- Systems with low-level granularity may lack control in **protecting data** from misuse
- System administrators are given better control of granularity at the URL level, shared directory level, and application level
- Different granularity is needed to control **information flows** within an application
- **Role-based Access Control (RBAC)** is a flow control model offering multiple levels of **control granularity**
  - Also called an **n-leveled RBAC (LnRBAC)**, as a level of granularity is controlled by a level of RBAC control
  - Handles and solves problems caused by abnormal program stopping
  - Controls method invocation, write accesses, and avoids Trojan horses

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Design Principles (Cont'd)

Fault Tolerance

- Strategy applied to **software design** (or system design) to permit system to continue functioning even in the presence of faults by enhancing its robustness

Fault Detection

- Closely linked to fault tolerance, used in **detecting faults** and producing appropriate responses of **system behavior**
- Examples include system monitors, safety monitors, built-in tests, loop-back tests, etc.

Fault Removal

- Removes faults during **design process**
- Examples include error detection, verification through inspection, built-in testing, correction functions, etc.

Fault Avoidance

- Avoids errors that contribute to system faults during the **development process**
- Examples include defensive programming, error minimization during design process, minimization of safety critical code, using appropriate SDLC techniques, etc.

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secure Design Principles (Cont'd)

Loose Coupling

- Procedures that **operate independently** from other procedures are called loosely coupled procedures
- Loose coupling describes the **exchange relationship** that occurs between two or more systems or organizations
- The concept loose coupling of systems is adopted when either source or destination machines are supposed to be **changed frequently**
- Loose coupling can be achieved in **web services** or **service-oriented architecture** by hiding the implementation details from the caller

Well-known Techniques that Create Loose Coupling

- Vendor and platform independent messages
- Coarse-grained, self-describing, and self-contained messages
- Well-defined interfaces
- Extensible versionable interfaces
- Constrained interfaces
- Stateless messaging
- Human readable strings (URIs) for service and instance addresses
- Stateless messaging where possible and appropriate
- Humans controlling clients where possible and appropriate
- Asynchronous exchange patterns where possible and appropriate

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Design Principles (Cont'd)

High Cohesion

- Procedures that perform a single function
- Cohesion is a measure of identifying the extent of strongly related functionalities in the **source code** of the single module
- A particular class is said to have high cohesion if the methods in that class perform **similar actions** in many aspects
- This high cohesively code increases the code **readability** and **reusability** without increasing complexity
- Procedures that are more reliable, easy to read, and maintain are usually **loosely coupled** and have **high cohesion**

Change Management and Version Control

- **Integrity** of system changes is controlled by a well-defined process called **configuration management**
- Allows all **stakeholders** to know upcoming changes
- Change management controls integration **costs** and **development**
- Required changes to code may cause **vulnerabilities**
- Anticipation of change requirements may **limit negative effects**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Threat Modeling

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

"It is not possible to design and develop a secure application until you know your threats."

Source: *msdn.microsoft.com*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Threat Modeling

- Threat modeling is a process of **identifying**, **analyzing**, and **mitigating the threats** to the application
- It is a structured approach that allows the developer to **rate the threats** based on the architecture and implementation of the application
- It is performed at the **design phase** of the secure development lifecycle
- It is an **iterative process** that starts from the design phase of the application and iterates throughout the application lifecycle until all possible **threats to the applications** are identified
- The output of threat modeling is a threats model exposing all the possible **threats** and **vulnerabilities** on an application

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

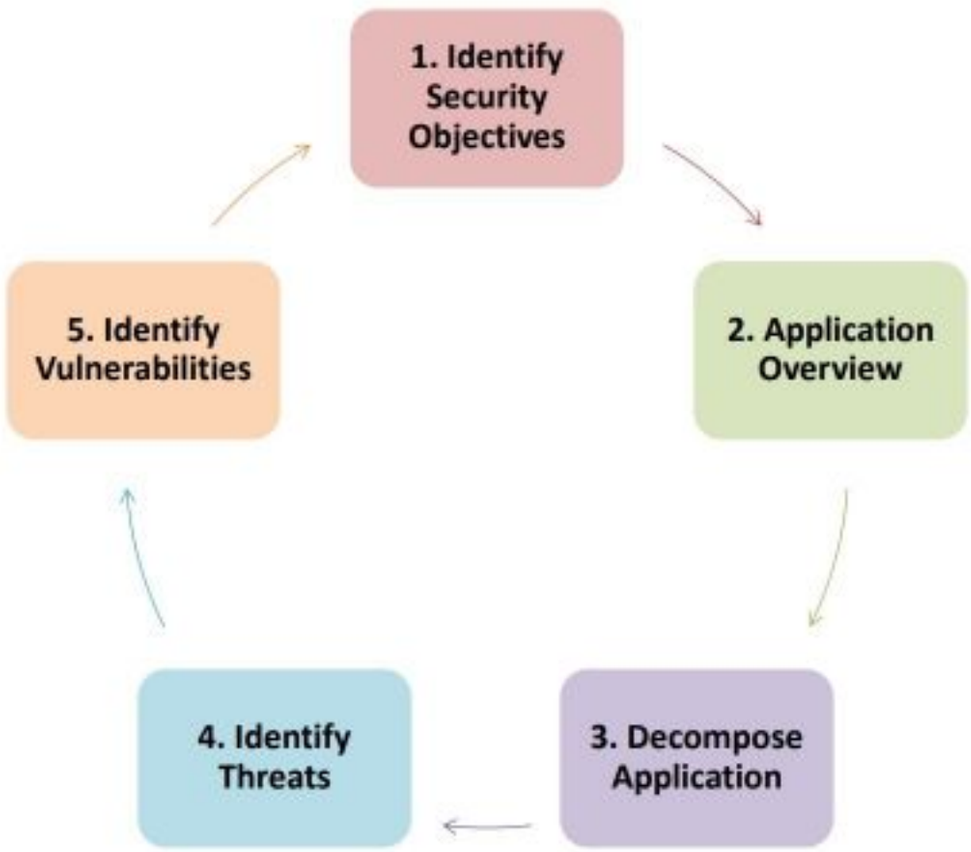
Threat Modeling Phases

Attack Surface Evaluation	Application is decomposed and its entry points are reviewed from an **attacker's** perspective
Threat Identification	Each entry points are then reviewed against **potential threats**
Impact Analysis	Impact of potential threats is calculated in terms of **risk**
Control Recommendations	**Security controls** are recommended to meet security objectives

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Threat Modeling Process

- **The steps involved in Threat Modeling process are as follows:**



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Identify the Security Objectives

- To proceed with creating threat model for an application, the **security objectives** of the application should be identified clearly
- These security objectives are **goals** and **constraints** that needs to be attain to maintain confidentiality, integrity, and availability of application and its data
- Security objectives are the subset of the application's **functional objectives**
- With security objectives clearly identified,
  - You can identify the areas which need more closer **attention** and where to focus your **efforts**
  - You can understand goals of potential attackers

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

How to Identify Security Objectives

- Security objectives are identified by conducting **brain storming sessions**, **questionnaires**, **storyboarding**, **etc.** with relevant stakeholders of the project

Sample Questionnaires	Sample Security Objectives
Does your application handles or processes sensitive data that you need to protect from unauthorized access? Identify such data that you need to protect from customers/clients Example of Sensitive data: user names, passwords, customer account details, personal information, financial details, transaction details, credit card numbers, bank details, etc.	Application should prevent unauthorized person from accessing confidential information of the customers
Does your application requires to comply with certain security policies, privacy laws, regulations, or standards Example: PCI-DSS, HIPPA, etc.	Application should meet the compliance requirements
Does organization wants application to satisfy certain quality of service requirements Example: availability, performance, etc.	Application should meet the quality of service requirements
Does application has any intangible assets that you need to protect For example: Organization's reputation, trade secrets, intellectual property, etc.	Application should protect the company's online business credibility

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Create an Application Overview

- The aim of creating application overview is to understand **what the application does**
- Creating an application overview Involves:
  - **Understanding** the application functionality
  - Understanding the **purpose** and the key **features** of the application
  - Identifying application **users** or **clients**
- **Create application overview with the help of these activities:**

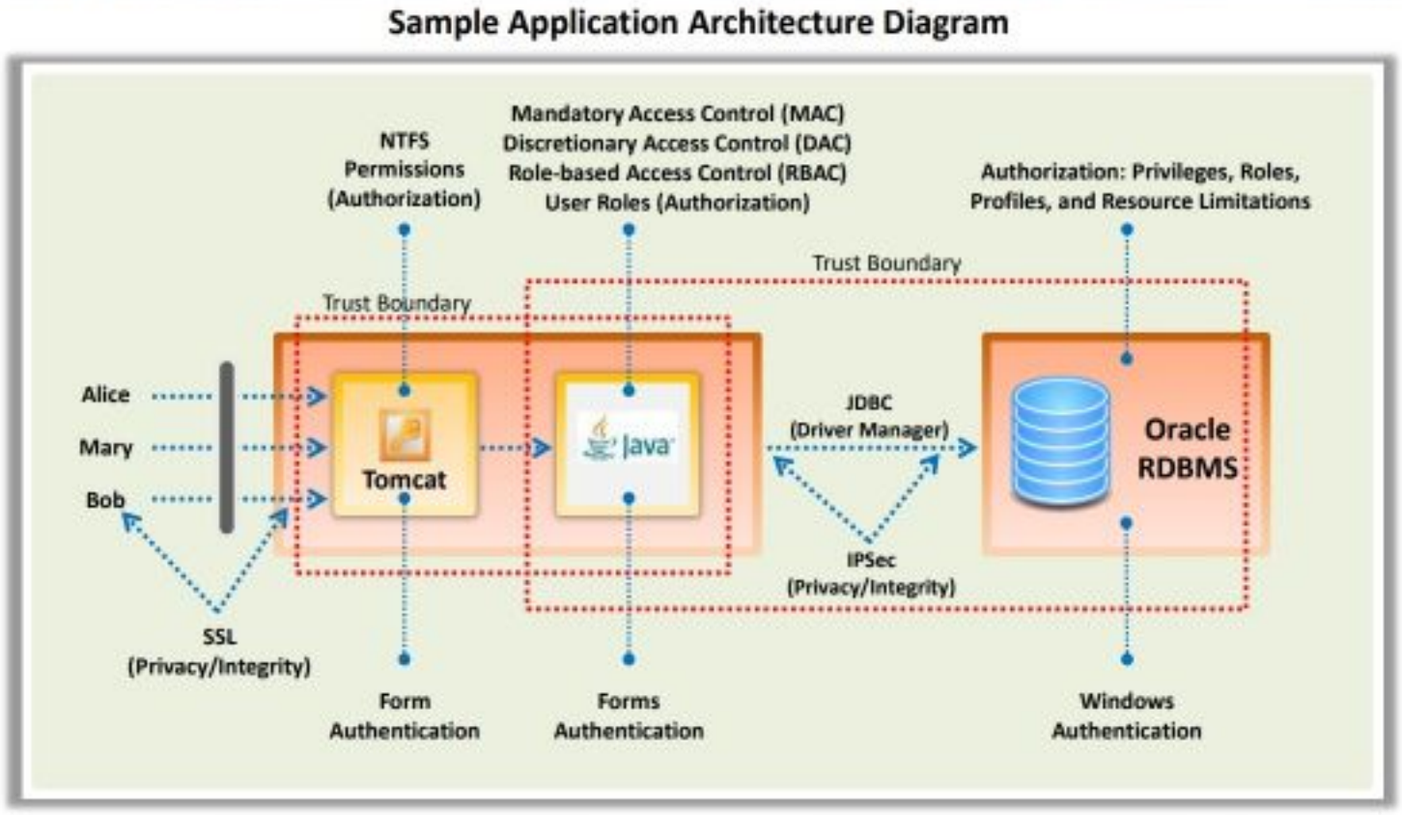
1. Draw the end-to-end deployment architecture
2. Identify various user roles
3. Identify use case scenarios
4. Identify technologies
5. Identify application security mechanisms

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Draw the End-to-End Deployment Architecture

- Draw a **high-level** diagram that describes the overall structure of the application and its subsystems in the deployment environment
- Identify end-to-end deployment topology, logical layers, key components and services, communication ports and protocols, etc.
- Add **authentication**, **authorization**, and **communication** mechanisms at broad level in the deployment diagram

Sample Application Architecture Diagram



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Identify Various User Roles

- Identify **who can do what** in the application
- Identify users or group of users with **higher privileges**
- Identify the **rights** of specific users

Examples of Roles
Anonymous Web User
User With Valid Login Credentials
User With Invalid Login Credentials
Librarian
Database Server Administrator
Website Administrator
Web Server User Process
Database Read User
Database Read/Write User

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Identify Use Case Scenarios

- Identify the **functionality** and **usage** of an application
- Identify **Create**, **Read**, **Update**, and **Delete** operations in the application
- Review following items to gain understanding of key usage of the application
  - Review existing documents
  - Review the use cases
  - Review requirements
- This will help you understand how the application should be **used** and how it can be **misused**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Identify Technologies

- Identify the **technologies** used to develop and deploy the application
  1. Operating systems
  2. Web server software
  3. Database server software
  4. Technologies used in the presentation, business, and data access layers
  5. Development languages
- This will help you identify **technology-specific threats**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Identify Application Security Mechanisms

- Identify and highlight **key elements** with respect to following application security mechanisms
  - ☐ Input and data validation
  - ☐ Authentication
  - ☐ Authorization
  - ☐ Configuration management
  - ☐ Sensitive data
  - ☐ Session management
  - ☐ Cryptography
  - ☐ Parameter manipulation
  - ☐ Exception management
  - ☐ Auditing and logging
- This will help you add **details** wherever necessary

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Decompose the Application

- Decompose the application to the level where it is possible to identify external entities, trust boundaries, data flow, entry points, and privileged code in the application and recognize the **areas of vulnerabilities** among them
- It helps to **conduct attack surface analysis**

- **Decompose application with the help of these activities:**
  - Prepare and Document Threat Model Information
  - Enumerate External Dependencies
  - Enumerate Entry Points
  - Enumerate Assets
  - Enumerate Trust levels
  - Data Flow Diagrams

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Prepare and Document Threat Model Information

- The following fields should be **documented** to begin with threat modeling

Application Name	The name of the application
Application Version	The current version of the application
Description	A high level description of the application
Document Owner	The name of the threat modeling document owner
Participants	The people involved in the threat modeling for the application
Reviewer	The name of the reviewer(s) of the threat model

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Example: Threat Model Information

Threat Model Information	
Application Version	1.0
Description	The college library website is the first implementation of a website to provide librarians and library patrons (students and college staff) with online services As this is the first implementation of the website, the functionality will be limited. There will be three users of the application: 1. Students 2. Staff 3. Librarians Staff and Students will be able to log in and search for books, and staff members can request books. Librarians will be able to log in, add books, add users, and search for books
Document Owner	John
Participants	Jason
Reviewer	Mathew

Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Identify the External Dependencies

- External dependency defines the application's dependency on **outside entities** such as servers, firewalls, security policies, OS, network, etc.
- These entities are the essential components of application but lay outside of the application's code
- Though these entities are outside of the application's control, they are still within the **control of the organization**
- Identifying such **external dependencies** may help to minimize the application's overall risk
- To find out external dependencies, analyze **different factors** on which application security will depend in a production environment
- For example
  - How the application will be deployed?
  - Will application is expected to run on the server?
  - Will there be firewall sitting before server?
- Document such external dependencies with ID and Description field

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

External Dependencies: Example

External Dependencies	
ID	**Description**
1	The college library website will run on a Linux server running Apache. This server will be hardened as per the college's server hardening standard. This includes the application of the latest operating systems and application security patches
2	The database server will be MySQL and it will run on a Linux server. This server will be hardened as per the college's server hardening standard. This will include the application of the latest operating system and application security patches
3	The connection between the Web Server and the database server will be over a private network
4	The Web Server is behind a firewall and the only communications is TLS

Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Identify the Entry Points

- Entry points are **interfaces** through which users input data or interact with the application
- A potential attacker may take advantages to such entry points to attack an application
- There can be multilayer entry points in the application; for example, external entry are exposed to the users and internal entry points exposed to subcomponents across the layers of the application
- Attacker can **bypass** first level of entry point and directly attack an internal entry point
- Document all the entry points including multi layer entry points; they should be documented with ID, Name, Description field
- **Typical entry points are:**
  - Login/authentication entry points
  - Admin interfaces
  - Inquiries and search functions
  - Data entry (CRUD) forms
  - Business workflows
  - Transactional interfaces/APIs
  - Operational command and monitoring interfaces/APIs
  - Interfaces with other applications/systems

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Entry Points: Example

Entry Points		
ID	**Name**	**Description**
1	HTTPS Port	The college library website will be accessible via TLS. All pages within the college library website are layered on this entry points
1.1	Library Main Page	The splash page for the college library website is the entry point for all users
1.2	Login Page	Students, faculty members and librarians must log in to the college library website before they can carry out of the use cases
1.2.1	Login Function	The login function accepts user supplied credentials and compares them with those in the database
1.3	Search Entry Page	The page used to enter a search query

Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Identify the Assets

- Application's asset is **resources/ items/areas** which has a value of attacker's interest
- They are essential **threats targets** which puts the application under risk of being attacked
- These assets can be **physical** and **abstract**
- For example,
  - Physical: data
  - Abstract: reputation, safety
- Assets can interact with other assets
- Identify all the assets of an application which needs to be protected from unauthorized access
- Document both physical and abstract assets of the application with its **ID**, **Name**, and **Description**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Assets: Example

Assets		
ID	**Name**	**Description**
1	Library Users and Librarians	Assets relating to student, faculty members, and librarians
1.1	User Login Details	The login credentials that a student or a faculty member will use to log into the college library website
1.2	Librarians Login Details	The login credentials that a librarian will use to log into the college library website.
1.3	Personal Data	The college library website will store personal information relating to the students, faculty members, and librarians
2	System	Assets relating to the underlying system
2.1	Availability Of College Library Website	The College Library website should be available 24 hrs. a day and can be accessed by all students, college faculty members, and librarians
2.2	Ability To Execute Code as a Web Server User	This is the ability to execute source code on the web server as a web server users

Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Assets: Example (Cont'd)

Assets		
ID	**Name**	**Description**
2.3	Ability To Execute SQL as a Database Read User	This is the ability to execute SQL select queries on the database, and thus retrieve any information stored within the College Library database
2.4	Ability To Execute SQL as a Database Read/Write User	This is the ability to execute SQL select, insert, and update quires on the database, and thus have read and write access to any information stored within the College Library database
3	Website	Assets relating to the College Library website
3.1	Login Session	This is the login session of a user to the College Library website. This user could be a student, a member of the college faculty, or a Librarian
3.2	Access to The Database Server	Access to the database server allows you to the administrator the database, giving you full access to the database user and all the data contain within the database
3.3	Ability to Create Users	The ability to create users would allow an individuals to create a new users on the system. These could be students users, faculty member users, and librarians users
3.4	Access to Audit Data	The audit data shows all audit-able events that occurred within the college library application by students, staff, and librarians

Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Identify the Trust Levels

- Trust level defines the **access rights** which applications should grant to the external entities
- Define the possible **roles** including set of privileges and trust levels assigned to the roles
- Document the **trust levels** with its ID, Name, and Description filed

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Trust Levels: Example

Trust Levels		
ID	**Name**	**Description**
1	Anonymous Web User	A user who has connected to the college library website but has not provided valid credentials
2	User with valid login credentials	A user who has connected to the college library website and logged in using valid credentials
3	User with invalid login credentials	A user who has connected to the college library website and is attempting to log in using invalid credentials
4	Librarian	The librarian can create users on the library website and view their personal information
5	Database Server Administrator	The database server administrator has read and write access to the database that is used by the college library website
6	Website Administrator	The website administrator can configure the college library website
7	Web Server User Process	This is the process/user that the web server executes code as and authenticates itself against the database server as.
8	Database Read User	The database user account used to access the database for read access
9	Database Read/ Write User	The database user account used to access the database for read and write access

Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Define Trust Levels to Entry points

- **Define the appropriate trust level required for each identified entry point**

Example:

Entry Points			
ID	Name	Description	Trust Levels
1	HTTPS Post	The college library will be only be accessible via TLS. All pages within the college library website are layered on this entry point	(1) Anonymous Web User (2) User with Valid Login Credentials (3) User with Invalid Login Credentials (4) Librarian
1.1	Library Main Page	The Splash page for the college library website is the entry point for all users	(1) Anonymous Web User (2) User with Valid Login Credentials (3) User with Invalid Login Credentials (4) Librarian
1.2	Login Page	Students, faculty members and librarians must log into the college library website before they can carry out any of the cases	(1) Anonymous Web User (2) User with Valid Login Credentials (3) User with Invalid Login Credentials (4) Librarian
1.2.1	Login Function	The login function accepts user supplied credentials and compares them with those in the database	(2) User with Valid Login Credentials (3) User with Invalid Login Credentials (4) Librarian
1.3	Search Entry Page	The page used to enter a search query	(2) User with Valid Login Credentials (4) Librarian

Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Define Trust Levels to Assets

- **Define the appropriate trust level required for each identified asset**

Example:

Assets			
ID	**Name**	**Description**	**Trust Levels**
1	Library Users and Librarian	Assets relating to students, faculty members, and librarians	
1.1	User Login Details	The login credentials that a student or a faculty member will use to log into the College Library website	(2) User with Valid Login Credentials (4) Librarian (5) Database Server Administrator (7) Web Server User Process (8) Database Read User (9) Database Read/Write User
1.2	Librarian Login Details	The login credentials that a Librarian will use to log into the College Library website	(4) Librarian (5) Database Server Administrator (7) Web Server User Process (8) Database Read User (9) Database Read/Write User
1.3	Personal Data	The College Library website will store personal information relating to the students, faculty members, and librarians	(4) Librarian (5) Database Server Administrator (6) Website Administrator (7) Web Server User Process (8) Database Read User (9) Database Read/Write User
2	System	Assets relating to the underlying system	

Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Define Trust Levels to Assets (Cont'd)

2.1	Availability of College Library Website	The College Library website should be available 24 hours a day and can be accessed by all students, college faculty members, and librarians	(5) Database Server Administrator (6) Website Administrator
2.2	Ability to Execute Code as a Web Server User	This is the ability to execute source code on the web server as a web server user	(6) Website Administrator (7) Web Server User Process
2.3	Ability to Execute SQL as a Database Read User	This is the ability to execute SQL select queries on the database, and thus retrieve any information stored within the College Library database	(5) Database Server Administrator (8) Database Read User (9) Database Read/Write User
2.4	Ability to Execute SQL as a Database Read/Write User	This is the ability to execute SQL. Select, insert, and update queries on the database and thus have read and write access to any information stored within the College Library database	(5) Database Server Administrator (9) Database Read/Write User
3	Website	Assets relating to the College Library website	
3.1	Login Session	This is the login session of a user to the College Library website. This user could be a student, a member of the college faculty, or a Librarian	(2) User with Valid Login Credentials (4) Librarian
3.2	Access to the Database Server	Access to the database server allows you to administer the database, giving you full access to the database users and all data contained within the database	(5) Database Server Administrator
3.3	Ability to Create Users	The ability to create users would allow an individual to create new users on the system. Theses could be students users, faculty member users, and librarian users	(4) Librarian (6) Website Administrator
3.4	Access to Audit Data	The audit data shows all audit-able events that occurred within the College Library application by students, staff, and librarians	(6) Website Administrator

Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Perform Application Modeling using Data Flow Diagrams (DFDs)

- The collected information such as **external entities, entry points, assets and trust level** will help in accurately modeling the application by using Data Flow Diagrams (DFDs)
- DFD will help in understanding how the data flows through the application and what happens to data while flowing
- Start with higher level DFD and go on creating lower DFDs to decompose the application into different process and their lower-level sub processes or functions
- The higher level DFD will help you understand overall the scope of the application while lower level DFDs will help you understand working of lower-level specific processes

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Perform Application Modeling using Data Flow Diagrams (DFDs) (Cont'd)

Different Components in DFD

External Entity — It is used to represents external entity which **interacts** with the **application from outside**

Process — It is used to represents **actions** or **function** to accomplish specific task

Multiple Process — It is used to represents **collection of sub processes**

Data Store — It is used to represents **locations** where data is stored

Data Flow — It is used to represents the **movement of data** within the application

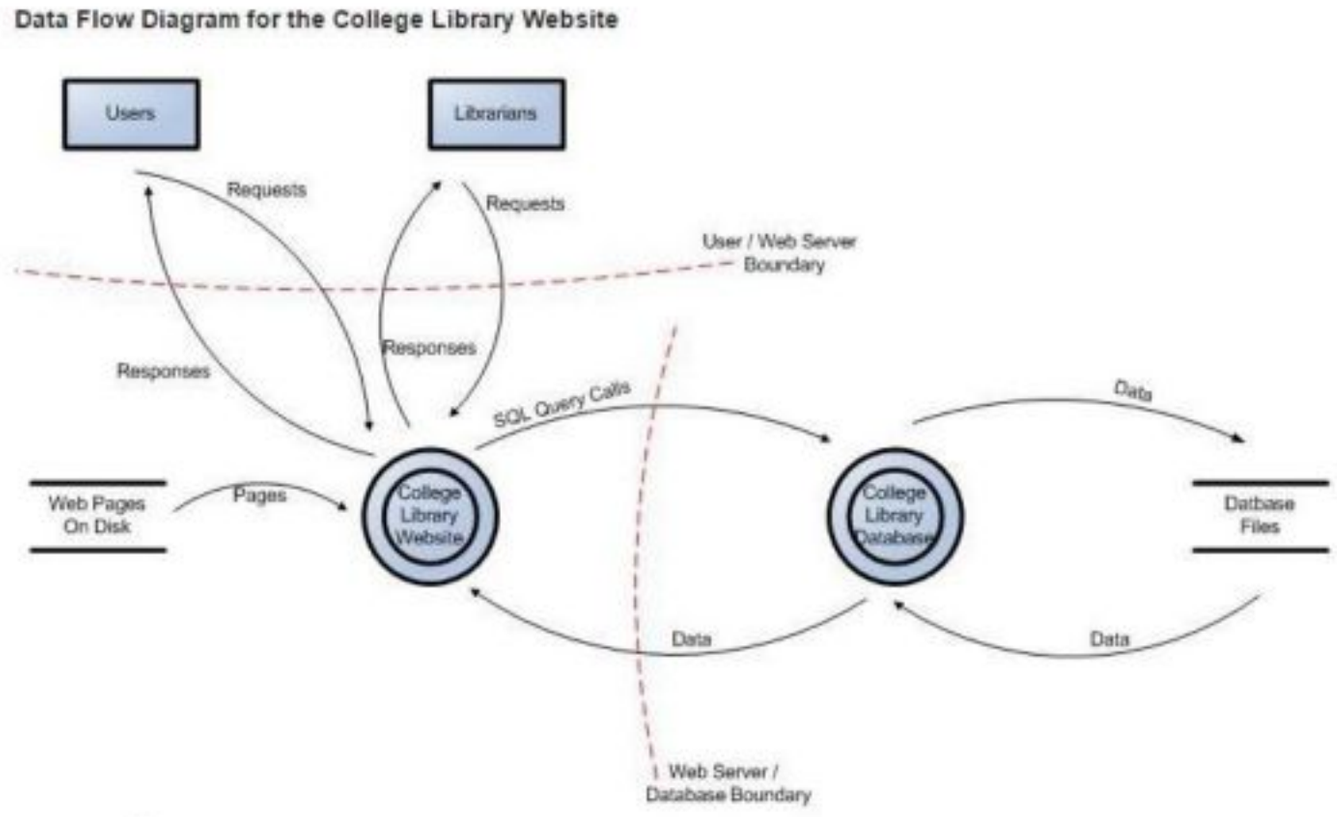
Privilege Boundary — It is used to represents **change of privilege levels**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Perform Application Modeling using Data Flow Diagrams (DFDs) (Cont'd)

- **Create Context level DFD to map the scope of the application**

Example:



Source: *https://www.owasp.org*

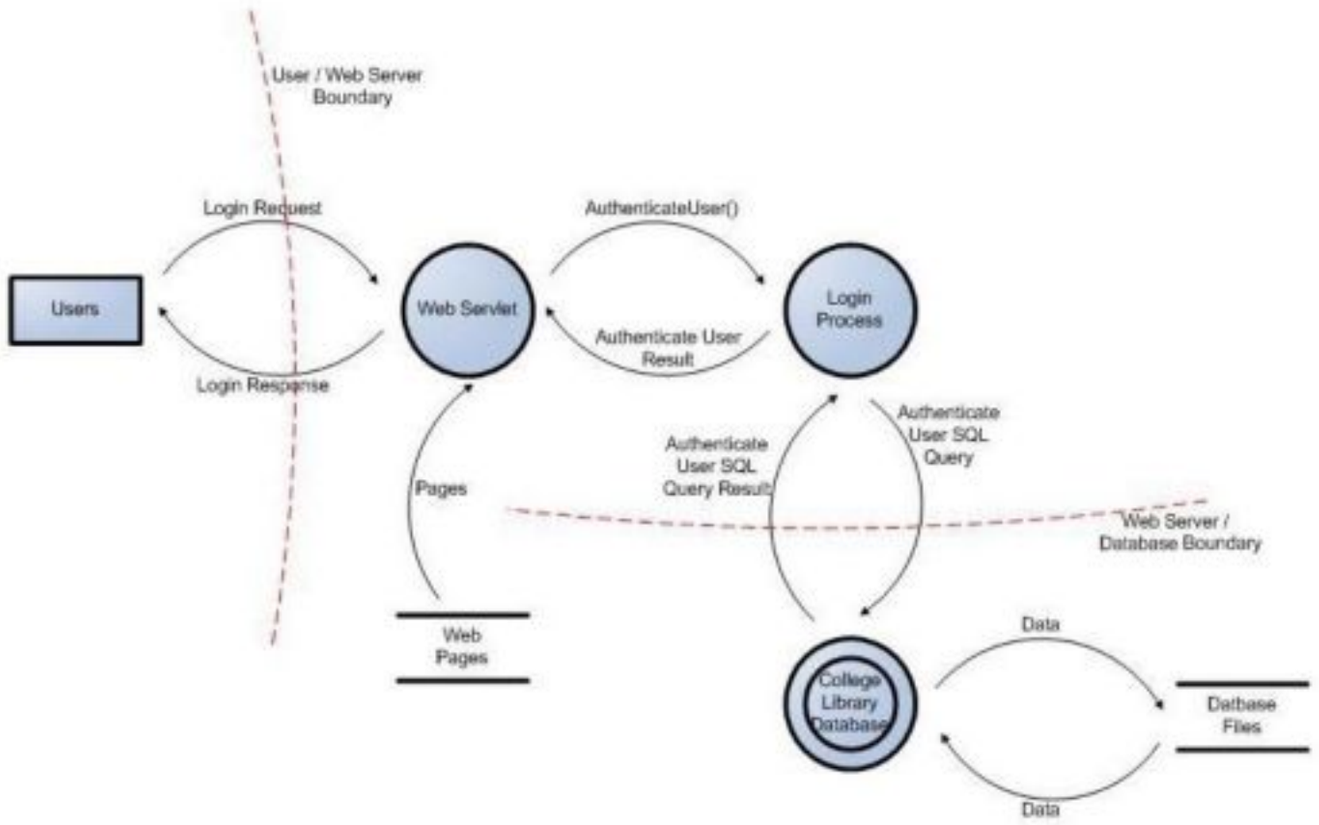
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Perform Application Modeling using Data Flow Diagrams (DFDs) (Cont'd)

- **Go on creating lower level DFD to depict specific process**

Example:



Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Determine the Threats: Identify the Goal of an Attacker and Create Threat Profile

- Once the DFD's are **created at lowest level**, you can identify all the external dependencies, entry/exits points, trust levels in an application
- It helps to easily identify what data need to be provided to specific process in DFD and corresponding **attacker's goal** to exploit the functionality of specific process
- This goal is used in DFD to determine the **threat path** in the application
- Identify all the possible **attacker's goals** and **vulnerabilities** which allows attackers to accomplish their goals

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Example: Attacker's Goal/Threat Profile and Vulnerabilities Associated

Example #1:

Goal: Attacker may steal user authentication credentials

Vulnerability:

- Insecure storage of credential at the client system
- Credentials are sent with GET parameters

Example #2:

Goal: Attacker may login with brute force attempt

Vulnerability :

- Use of Improper input validation and weak password

Example#3:

Goal: Attacker may launch a denial of service attack

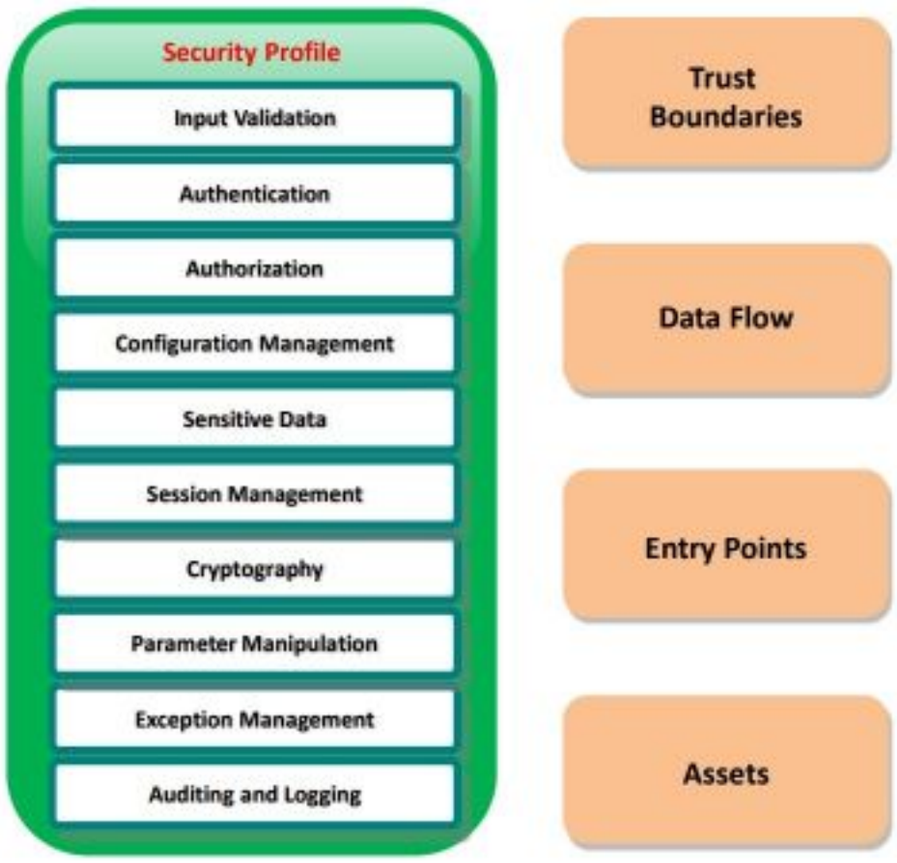
Vulnerability:

- Lack of account lockout functionality
- Absence of CAPTCHA

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Determine the Threats: Create a Security Profile

- To create **security profile** for an application, focus on:
  - Various **design** and **implementation approaches** of an application which can be most susceptible to vulnerabilities
  - Various **areas of applications** which can be most susceptible to vulnerabilities
- Document the **security profile** for application depending on the area of vulnerabilities



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Determine the Threats: Create a Security Profile (Cont'd)

- **Following considerations help you to identify vulnerabilities in design and implementation of your application**

Category	Considerations
Input Validation	Is all input data validated? Could an attacker inject malicious commands or data in the input fields? Is data validated as it is passed between separate trust boundaries (by the recipient entry point)? Can data in the database be trusted?
Authentication	Are credentials secured if they are passed over the network? Are strong account policies used? Are strong passwords enforced? Are you using certificates? Are password verifiers (using one-way hashes) used for user passwords?
Authorization	What gatekeepers are used at the entry points of the application? How is authorization enforced at the database? Is defense in depth strategy used? Do you fail securely and only allow access upon successful confirmation of credentials?
Configuration Management	What administration interfaces does the application support? How are they secured? How is remote administration secured? What configuration stores are used and how are they secured?
Sensitive Data	What sensitive data is handled by the application? How is it secured over the network and in persistent stores? What type of encryption is used and how are encryption keys secured?

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Determine the Threats: Create a Security Profile (Cont'd)

Category	Considerations
Session Management	How are session cookies generated? How are they secured to prevent session hijacking? How is persistent session state secured? How is session state secured as it crosses the network? How does the application authenticate users with the session store? Are credentials passed transmitted and are they maintained by the application? If so, how are they secured?
Cryptography	What algorithms and cryptographic techniques are used? How long are encryption keys and how are they secured? Does the application put its own encryption into action? How often are keys recycled?
Parameter Manipulation	Does the application detect tampered parameters? Does it validate all parameters in form fields, ViewState, cookie data, and HTTP headers?
Exception Management	How does the application performs error handling? Are exceptions ever allowed to propagate back to the client? Are generic error messages that do not contain exploitable information used?
Auditing and Logging	Does your application audit activity across all tiers on all servers? How are log files secured?

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Identify the Threats

1. To identify threats, they need to be broadly categorized into **different possible categories**
2. This threat categorization provides structured way to **systematically identify different threats** on the application
3. Threat identification process uses **STRIDE model** to categorize threats and examine each aspect of the security profile of the application

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Identify the Threats: The STRIDE Model

- The STRIDE model categorizes the application threats based on the **goals** and **purpose of the attack** which helps developer to **develop a security strategy**
- It also includes the **countermeasures** for all threat categories
- **This model describes the following threat categories:**

Desired Property	Threats	Description
Authentication	Spoofing	Unauthorized access to a system by using a false identity
Integrity	Tampering	Code and data modification without authorization
Non-repudiation	Repudiation	Ability of users to claim that they do not perform some actions against the application
Confidentiality	Information disclosure	Unwanted exposure of private data to unauthorized users
Availability	Denial of service	Ability to deny a system or application unavailable to the legitimate users
Authorization	Privilege escalation	Ability of user with limited privileges to elevate their privileges with an application without authorization

Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Example: Threat Categorized and Identified using STRIDE

STRIDE Threat List	
Type	**Examples**
Spoofing	Threat action aimed to illegally access and use another user's credentials, such as username and password
Tampering	Threat action aimed to maliciously change/modify persistent data, such as persistent data in a database, and the alteration of data in transit between two computers over an open network, such as the internet
Repudiation	Threat action aimed to perform illegal operations in a system that lacks the ability to trace the prohibited operations
Information disclosure	Threat action to read a file that one was not granted access to, or to read data in transit
Denial of Service	Threat aimed to deny access to valid users, such as by making a web server temporarily unavailable or unusable
Elevation of privilege	Threat aimed to gain privilege access to resources for gaining unauthorized access to information or to compromise a system

Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Determine Countermeasures and Mitigation Security Controls

- **Identify the set of controls and countermeasures to prevent identified threat**

Desired Property	Threats	Countermeasures/Controls
Authentication	Spoofing	Use strong authentication mechanisms Avoid storing credentials in clear text Do not pass sensitive data in clear text over networks
Integrity	Tampering	Use strong authentication mechanisms Secure authentication tokens with encrypted communication channels Use digital signature
Non-repudiation	Repudiation	Identify malicious behavior Create secure audit trails Use digital signature
Confidentiality	Information disclosure	Use strong authorization and encryption mechanism Avoid storing credentials in clear text
Availability	Denial of service	Use bandwidth controlling techniques
Authorization	Privilege escalation	Use limited privileged service accounts

Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Document the Threats

- A standard template that describes several **threat attributes** including threat description and threat target are used to document the threat
- The **risk rate** is left blank as it is to be filled in during the final risk rating step

A Sample Template for Documenting the Threats

Threat Description	Attacker obtains Authentication Credentials by Monitoring the Network
Threat Target	Web application user authentication process
Risk	
Attack Techniques	Use of network monitoring software
Countermeasures	Use SSL to provide encrypted channel

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Rating the Threats

1. This step rates the threats based on the **risks they possess**
2. Threat resolution is prioritized based on the **risk ratings**
3. The formula for calculating risk is:
   RISK = PROBABILITY * DAMAGE POTENTIAL

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Rating the Threats: DREAD Model

DREAD Model

- DREAD stands for damage potential, reproducibility, exploitability, affected users, and discoverability
- DREAD model is used to rate the various security threats on the application by **calculating risks of each threats**
- The severity levels are assigned to various threats on the application so as to **mitigate these threats as per their severity**

	Rating	High (3)	Medium (2)	Low (1)
D	Damage Potential	The attacker can subvert the security system; get full trust authorization run as administrator; upload content	Leaking sensitive information	Leaking trivial information
R	Reproducibility	The attack can be produced every time and does not require a timing window	The attack can be reproduced, but only with a timing window and a particular race situation	The attack is very difficult to reproduce even with knowledge of the security hole
E	Exploitability	A novice programmer could make the attack in a short time	A skilled programmer could make the attack then repeat the steps	The attacker requires an extremely skilled person and in-depth knowledge every time to exploit
A	Affected Users	All users, default configuration key customers	Some users, non-default configuration	Very small percentage of users, obscure feature affects anonymous users
D	Discoverability	Published information explains the attack. The vulnerability is found in the most commonly used feature and is very noticeable	The vulnerability in a seldom-used part of the product and only a few users should come across it. It would take some thinking to see malicious use	The bug is obscure and it is unlikely that users will work out damage potential

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Rating the Threats: DREAD Model (Cont'd)

Sample DREAD Rating Table

Threat	D	R	E	A	D	Total	Rating
Attacker obtains authentication credentials by monitoring the network	3	3	2	2	2	12	High
SQL commands injected into application	3	3	3	3	2	14	High

A Sample Template for Documenting the Threats after Risk Rating

Threat Description	Attacker obtains Authentication Credentials by Monitoring the Network
Threat target	Web application user authentication process
Risk rating	High
Attack techniques	Use of network monitoring software
Countermeasures	Use SSL to provide encrypted channel

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Application Architecture

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Design Secure Application Architecture

1. A typical web application architecture comprises of three tiers i.e. **web**, **application** and **database**
2. Security at one tier is not enough as **attacker** can breach the security of another tier to compromise the application
3. Design web application architecture with **defense-in-depth** principle i.e. providing security at each tier of the web application
4. A multi-tiered security include proper input validation, **database layer abstraction**, server configuration, proxies, web application firewalls, data encryption, OS hardening, and so on

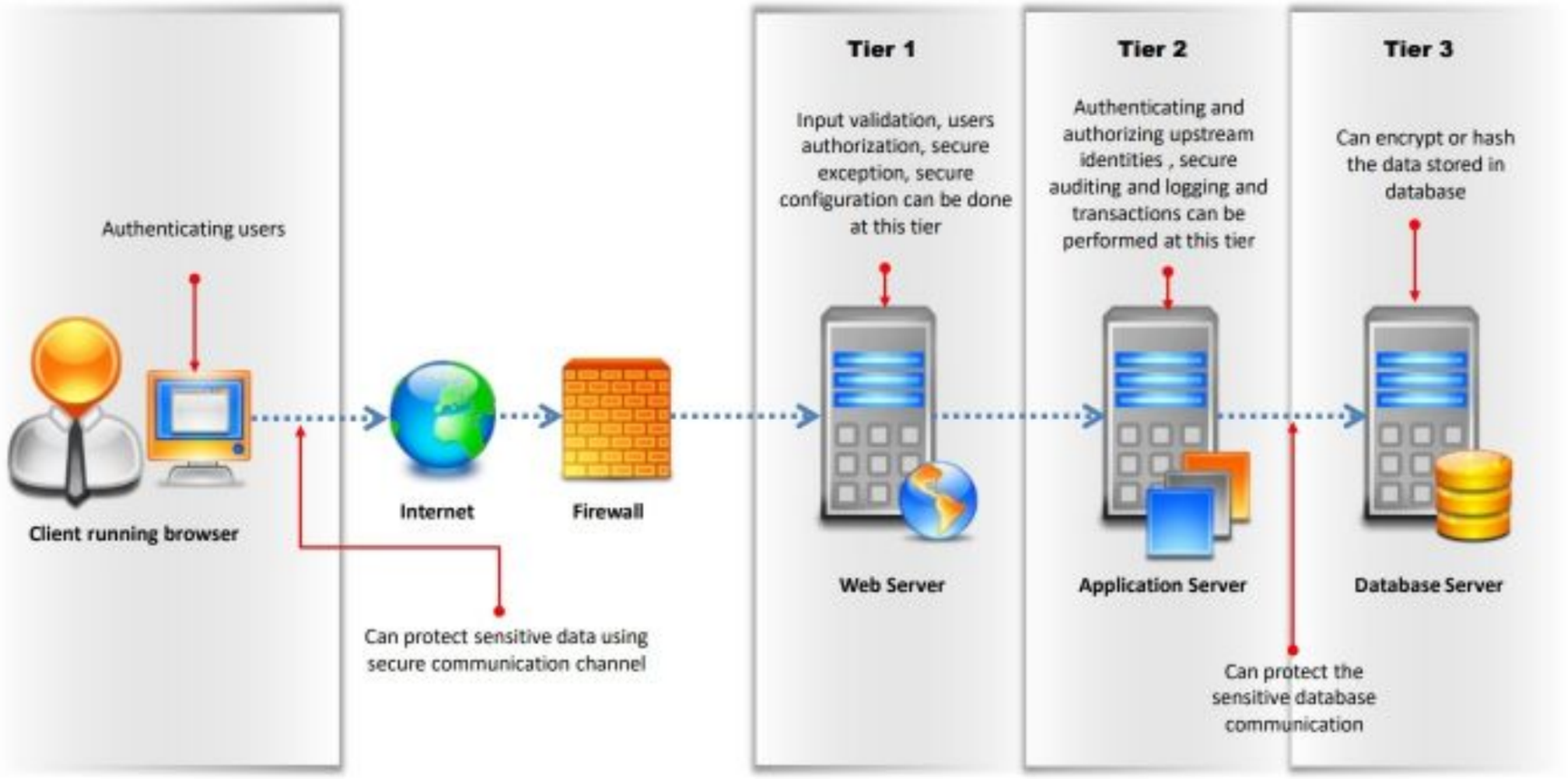
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Design Secure Application Architecture (Cont'd)

- **Applying multiple layer security in application architecture design makes application robust and secure**



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Module Summary

- Security negligence at design and architecture phase may lead to vulnerabilities that are most difficult to detect and expensive to fix in production
- Secure design of an application is more straightforward once the security requirements are identified
- Secure design principles are the state of practices or guidelines which should be enforced on the developers to follow during development phase
- Threat modeling is a process of identifying, analyzing, and mitigating the threats to the application
- A multiple tired security include proper input validation, database layer abstraction, server configuration, proxies, web application firewalls, data encryption, OS hardening, etc.

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

JAVA

C | ASE

Certified | Application Security Engineer



Module 04

Secure Coding Practices for Input Validation

@ErrorCatchers

This page is intentionally left blank.

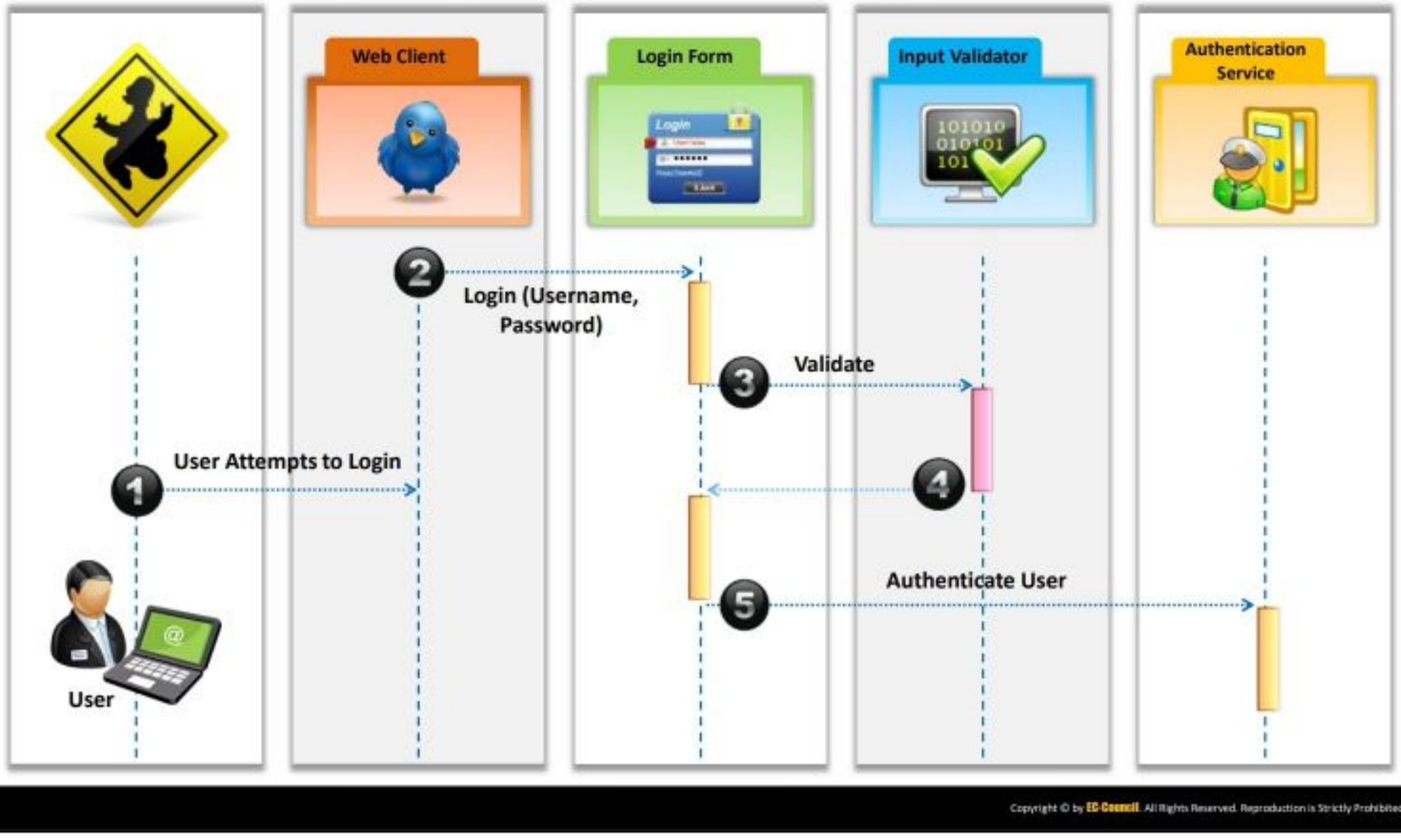
@ErrorCatchers

Module Objectives

1. Understand the Need of Input Validation
2. Discuss Data Validation Techniques
3. Discuss Data Validation in Strut Framework
4. Discuss Data Validation in Spring Framework
5. Discuss Common Input Validation Errors
6. Learn Common Secure Coding Practices for Input Validation

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Input Validation Pattern



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Validation and Security Issues

- Humans intentionally or unintentionally give invalid inputs that can pose computer **security risks** to applications and organizations
- A web application surely needs to check various **mandatory application fields** for correct expected format else it results in altered control flow, arbitrary control of a resource, or arbitrary code execution
- Java **action forms** provide input data validation in the source code, providing error message in case of invalid data input

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Impact of Invalid Data Input

- Input data validation is most important for any software application because of the dangers posed by **tainted input** at various levels of its execution
- An application that does not validate the input data is prone to **risks** that are **critical** to application security

Risks of Improper Data Validation

Availability (Technical Impact)

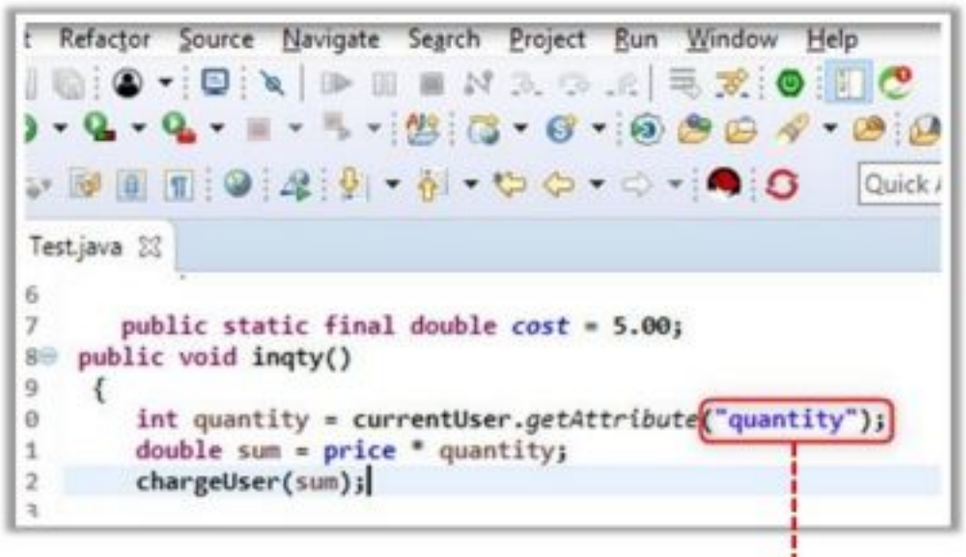
- A system can be DOS attacked, **crashed**, **exited**, **restarted** or resources like CPU, memory are consumed

Confidentiality

- Attackers can steal, read or change the data such as **memory**, **files** and **directories**

Integrity

- Hackers can illegitimately modify the **control flow**, **execute arbitrary commands**, and modify the data and memory

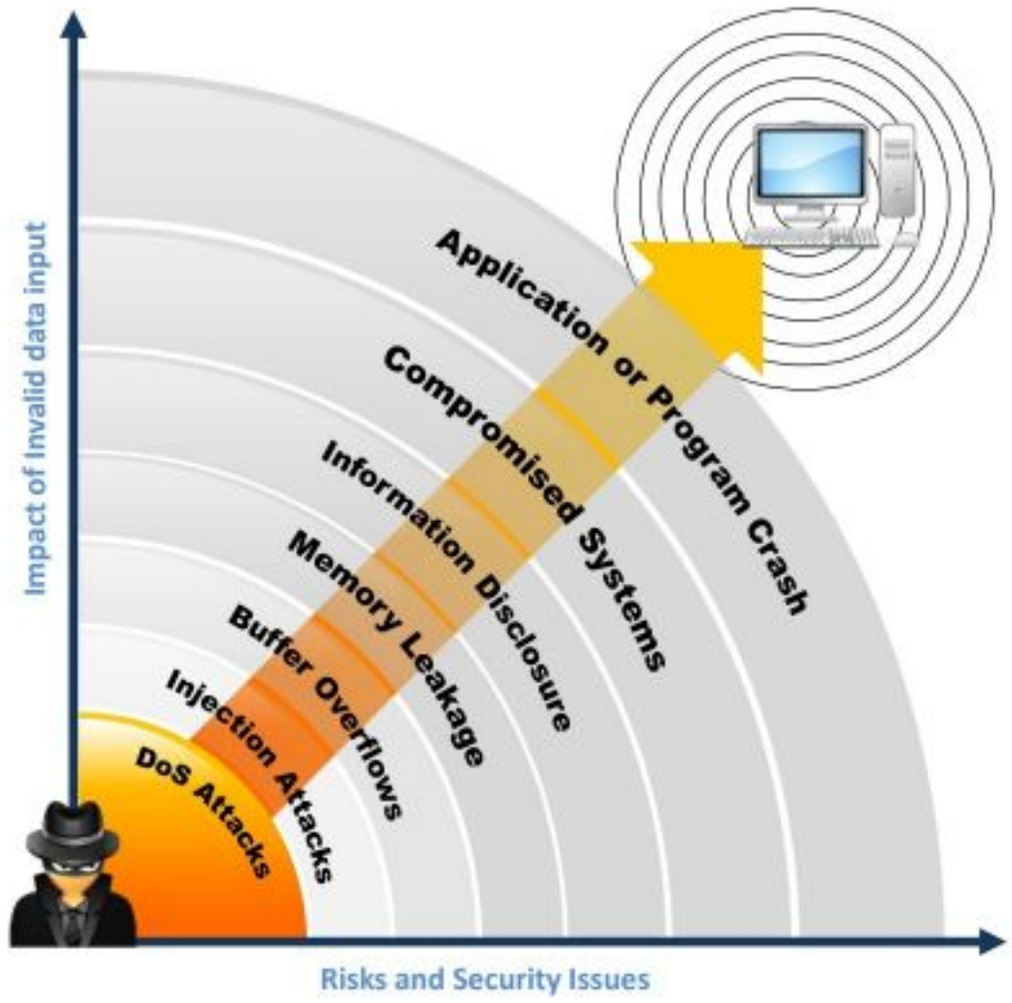


In this code, an e-commerce application price of a particular article is constant, where as the quantity is not; a hacker can even insert negative values for the quantity resulting in crediting his account instead of debiting on buying the good

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Impact of Invalid Data Input (Cont'd)



Target	Type of Attacks
Browsers	HTML Splitting, XFS, XSS
Data Repositories	LDAP Injection, SQL Injection
Server Attacks	Xpath Injection, XML Injection, XXE Injection
Application, Server and OS Attacks	Buffer Overflows and File uploads

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Data Validation Techniques

1. Data validation techniques include various methods used in **preventing attackers** from taking advantage of the vulnerabilities arising due to **invalid inputs**
2. Canonicalization of input or encoding of the input that can be easily interpreted by the system is always **safe**

Character set that mostly cause trouble are:

- Characters that have special meaning
- Normal control characters (values less than 32)
- Characters with values higher than 127
- Metacharacters

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Data Validation Techniques (Cont'd)

- **Data validation techniques include some of the basic general strategies such as:**

Encode Known Bad

- Encoding the known bad entries within the code

Accepting Exact Match

- Accepting a finite list of only known values

Reject Known Bad

- Do not let known bad entries to be entered into the system

Known Good

- If a finite list is not available, then use known good approach

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Input Validation using Frameworks and APIs

Java Framework

- Java framework allows the control of user behavior in Java and helps save time by eliminating the creation of **test scripts**
- This is possible by embedding Java source code into the framework

Java API

- Java API contains a **set of classes** in Java development environment
- It allows programmers to perform common tasks such as file transfer, networking, and data structures

Need for Input Validation

- Every application should be designed in such a way that it can ensure by itself that only valid data is inserted through **predetermined parameters**
- Any attempt to insert or update malicious data should be **detected** and **rejected** by the application



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Open Source Validation Framework for Java

- **Following are commonly used data validation libraries/framework**

- **Commons Validator**: Supports both server side and client side validation
- **spring-modules-validation**: It contains many tools and add-ons and it is based on validator specification
- **OVal**: It validates Java objects as per user request. It checks the parameters on calling the constructor
- **iScreen**: It validates Java objects via configuration. It is used for validating Java beans
- **jReform**: It is mainly used for HTML form and it automatically validates inputs
- **JValidate**: This framework supports Java 1.4 and above. It implements validations with annotations
- **JValidations**: Validation is done by the object itself without any external classes

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Servlet Filters

- Input validation through servlet filters in Java web application is effective due to **minor modifications** needed for input validation and servlets filters are **centralized** in nature
- Input validation in servlets is through **multipart encoded content** by handling multipart requests
- Applications should not rely only on one form of defense, i.e. servlet filters alone; **security** is a multi layered approach
- Modern frameworks facilitate input validation in the application itself
- Servlets can be configured on an application if it does not require multipart requests where **automatic multipart requests** should be disabled

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Validation Filters for Servlet

- The validation filter is executed before processing the request
- Validation filters **Error-Message** attribute stores error messages and displays the same when validation fails
- Filter mapping can be done with the help of **@webFilter** annotation
- @webFilter when applied on a url, the filter is invoked before invocation of url pattern eg. **/Information**
- If the data fails to validate then it redirects to the **error page** specified using **RequestDispatcher**

MyValidationFilter.java

```
1  package com.ec.web;
2
3  import java.io.IOException;
4  import javax.servlet.Filter;
5  import javax.servlet.FilterChain;
6  import javax.servlet.FilterConfig;
7  import javax.servlet.RequestDispatcher;
8  import javax.servlet.ServletException;
9  import javax.servlet.ServletRequest;
10 import javax.servlet.ServletResponse;
11 import javax.servlet.annotation.WebFilter;
12 import org.apache.commons.validator.routines.EmailValidator;

14 @WebFilter(filterName="MyValidationFilter", urlPatterns="Information")
15 public class MyValidationFilter implements Filter {

17     public MyValidationFilter() {    }
18
19     public void destroy() {
20     }
21     public void doFilter(ServletRequest request, ServletResponse response, FilterC
22
23
24 String Itserpg = "ErrorPag.jsp";
25 String MyuserName = request.getParameter("Myusername");
26 String Myemail = request.getParameter("Myemail");
27 boolean valid = EmailValidator.getInstance().isValid(Myemail);
28
29 if (MyuserName == null || "".equals(MyuserName)  || Myemail == null || "".equals(M
30 {
31 request.setAttribute("Error-Message", "One or both fields are empty");
32 RequestDispatcher rd = request.getRequestDispatcher(Itserpg);
33 rd.include(request, response);
34 } else if (!valid)
35 {
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Data Validation using OWASP ESAPI

- OWASP ESAPI is free open source API for securing web applications
- It can be used to **implement** and **enhance** security of existing applications
- It contains:
  - A set of security control interfaces
  - Reference implementation for security control
  - Custom implementations for each security control

Example : Input Validation using OWASP ESAPI

- Package **org.owasp.esapi.reference.validation** provides following **data validation classes** :
  - **BaseValidationRule**: Validating untrusted source
  - **CreditCardValidationRule**: Validating credit card no.
  - **DateValidationRule**: Validating date
  - **HTMLValidationRule**: Validating HTML to protect from XSS attack
  - **IntegerValidationRule**: Validating integers
  - **NumberValidationRule**: Validating number input format
  - **StringValidationRule**: Validating strings

*ValidateUser.java

```
2
3  import org.owasp.esapi.ESAPI;
4  import org.owasp.esapi.Validator;
5  public class ValidateUser {
6  public void uinfo()
7  {
8      Validator instance = ESAPI.validator();
9      Assert.assertTrue(instance.isValidInput("itstest", "rrdham@mail.com", "Email", 100, false));
10 }
11 }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Data Validation: Struts Framework

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Struts Validator

- The **Struts validator framework** is used to validate input on both the user browser and the server side
- The Struts framework makes use of a utility called **commons validator** for validating input
- The common validator gives the user a central area for data validation

The validator plug-in is mapped in the struts-config.xml file as follows:



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Struts Validation and Security

I. Struts validation is done to prevent attacks caused through **unchecked input**

II. All the inputs should be checked using the **struts validator** before it is processed by the application

III. Each and every field included in the form should be validated in the **corresponding validation form**

IV. The absence of validation for a **single field** may allow attackers to exploit the application

V. Vulnerabilities in input validation may lead to **buffer overflow attacks**, **cross-site scripting**, **SQL injection**, etc.

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Data Validation using Struts Validator

- **The following principles should be implemented in code when validating data using the Struts Validator**
  - Avoid **duplicate validation** forms in the validation xml file
  - Implement **struts validator** class
  - Enable the struts validator in the action form mapping
  - Check for similar number of fields in **action form** and **validation form**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Avoid Duplication of Validation Forms

Validation forms in struts are used to **validate** the **input fields**

Validation forms with the same name should be avoided when mapping in validation xml files

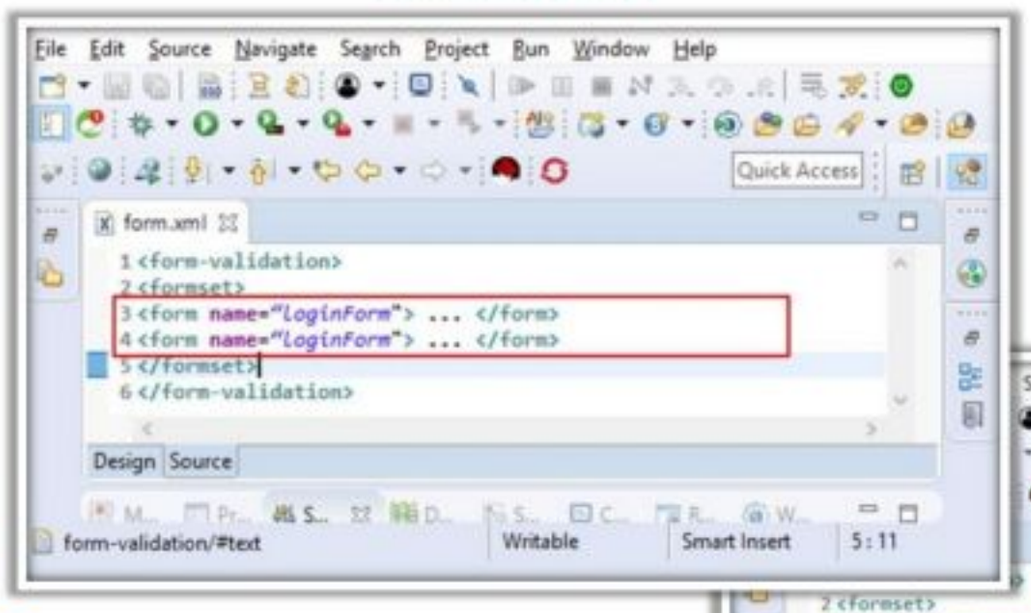
If two forms are given the same name, it indicates that the validation logic is incorrect

When similar names are given, the **struts validator** selects one form in a random manner and discards the other form

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

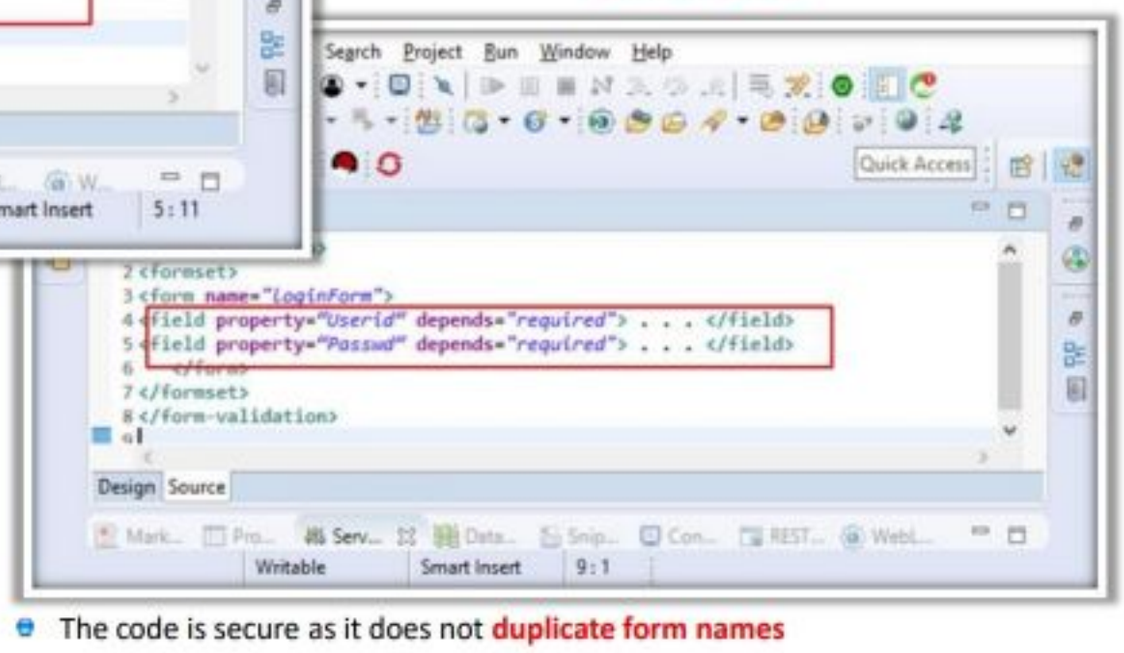
Secure and Insecure Struts Validation Code

Vulnerable Code



- The vulnerable code should be avoided in **validation** of **xml file**

Secure Code



- The code is secure as it does not **duplicate form names**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Struts Validator Class

- The **struts form** should extend one of the following validator classes to implement struts validator
  - ValidatorForm
  - ValidatorActionForm
  - DynaValidatorActionForm
  - DynaValidatorForm
- The method **validate()** can be implemented in the user's application through the use of these classes
- The validation of form fields are done under the **validate()** method
- The validator form that defines a **validate()** method should call **super.validate()** function
- The validator forms derived from **ActionForm** and **DynaActionForm** classes cannot use the struts validator

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secure and Insecure Code for Struts Validator Class

Vulnerable Code

MyLoginForm.java

```
1  package com.valila.form;
2
3  import javax.servlet.http.HttpServletRequest;
4
5  import org.apache.struts.action.ActionErrors;
6  import org.apache.struts.action.ActionMapping;
7  import org.apache.struts.action.ActionMessage;
8  import org.apache.struts.validator.ValidatorForm;
9
10 public class MyLoginForm extends ValidatorForm {
11     /**
12      *
13      */
14     private static final long serialVersionUID = 1L;
15     private String empname;
16     public String getEmpname() {
17         return empname;
18     }
19     public void setEmpname(String empname) {
20         this.empname = empname;
21     }
22     public String getEmpemail_id() {
23         return empemail_id;
24     }
25     public void setEmpemail_id(String empemail_id) {
26         this.empemail_id = empemail_id;
27     }
28     private String empemail_id;
29
30     public MyLoginForm()
31     {
32         super();
33     }
34     public ActionErrors validate(ActionMapping map, HttpServletRequest request) {
35         ActionErrors act_err = new ActionErrors();
36         if (getEmpname() == null || getEmpname().length() < 1) {
37         act_err.add("name", new ActionMessage("error.name.required"));
38         }
39         return act_err;
40         }
```

- The code is vulnerable as it does not call **super.validate()** function

Secure Code

MyLoginForm.java

```
1  package com.valila.form;
2
3  import javax.servlet.http.HttpServletRequest;
4
5  import org.apache.struts.action.ActionErrors;
6  import org.apache.struts.action.ActionMapping;
7  import org.apache.struts.action.ActionMessage;
8  import org.apache.struts.validator.ValidatorForm;
9
10 public class MyLoginForm extends ValidatorForm {
11     /**
12      *
13      */
14     private static final long serialVersionUID = 1L;
15     private String empname;
16     public String getEmpname() {
17         return empname;
18     }
19     public void setEmpname(String empname) {
20         this.empname = empname;
21     }
22     public String getEmpemail_id() {
23         return empemail_id;
24     }
25     public void setEmpemail_id(String empemail_id) {
26         this.empemail_id = empemail_id;
27     }
28     private String empemail_id;
29
30     public MyLoginForm()
31     {
32         super();
33     }
34     public ActionErrors validate(ActionMapping map, HttpServletRequest request) {
35         ActionErrors act_err = super.validate(map, request);
36         if (getEmpname() == null || getEmpname().length() < 1) {
37         act_err.add("name", new ActionMessage("error.name.required"));
38         }
39         return act_err;
40         }
```

- The code calls **super.validate()** function

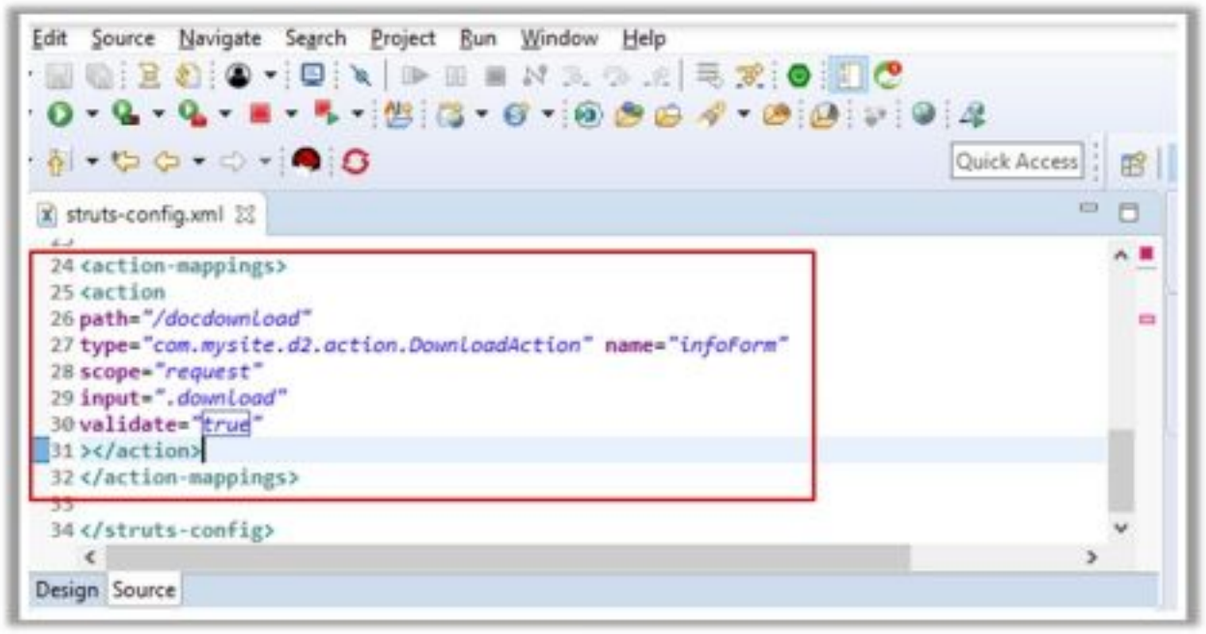
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

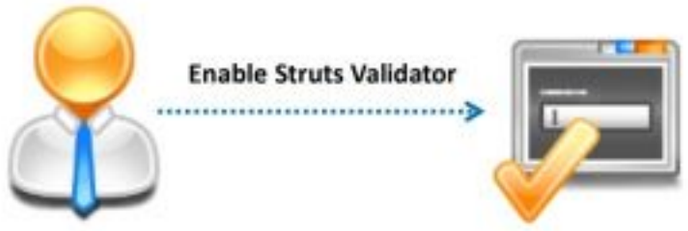
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Enable the Struts Validator

- The action form mapping for the **validate()** method should always be given the value **true**
- The form disables the struts validator if the **action form mapping** is **disabled**

The following code shows how to enable action form mapping

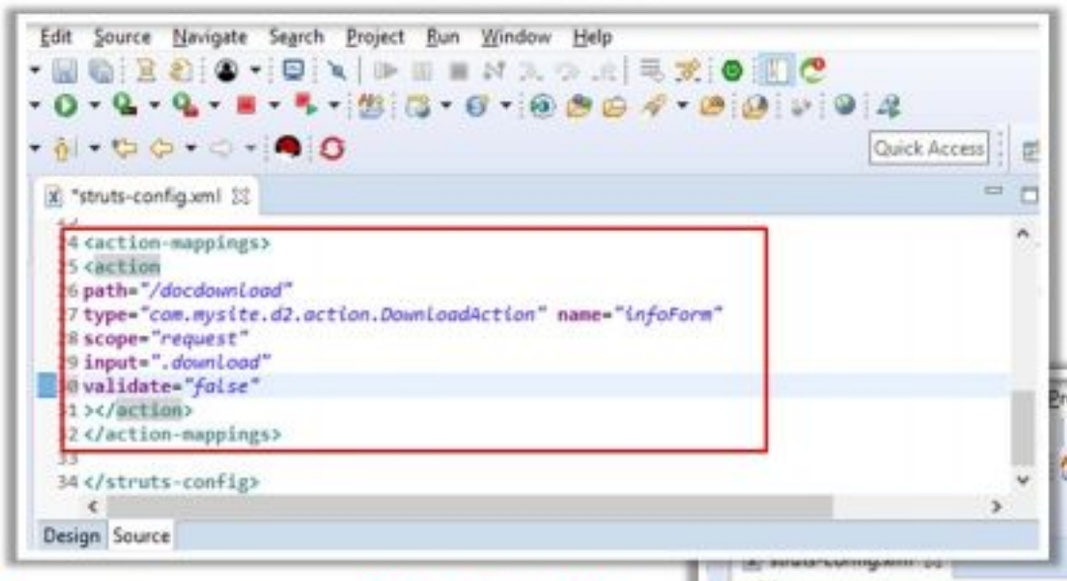




Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

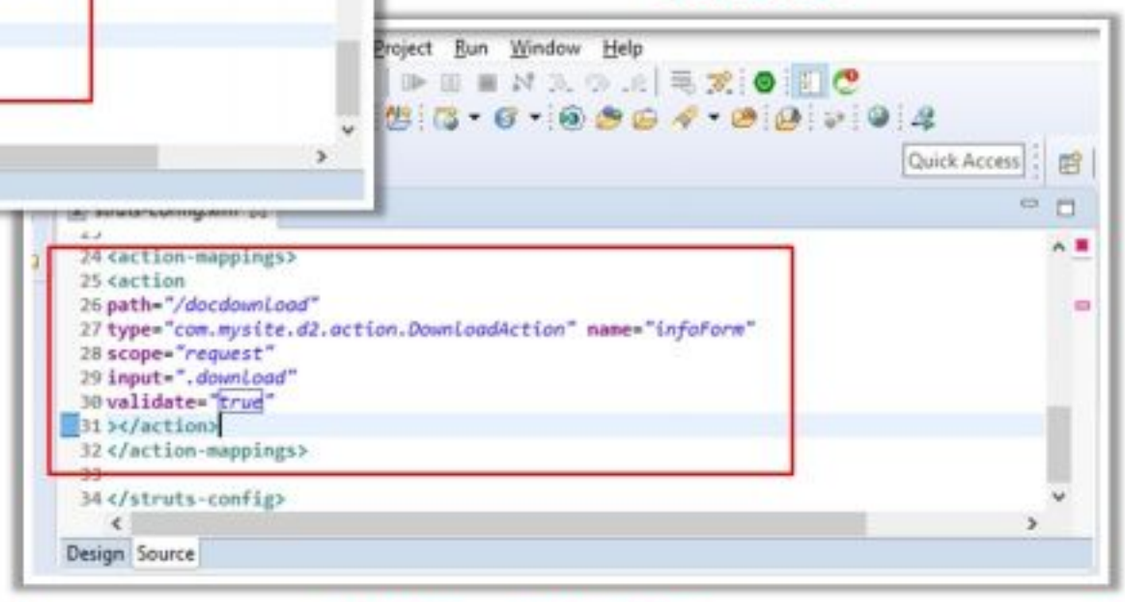
Secure and Insecure Struts Validator Code

Vulnerable Code



- The code is vulnerable as it **deactivates** the **struts validator** as the validate parameter is set to **false**

Secure Code



- The code is secure as it **deactivates** the **struts validator** as the validate parameter is set to **true**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Struts 2 Framework Validator

- Struts 2 implements **XWork framework** for input validations
- Struts 2 validation framework separates the **actual validation** logic and **application code**
- It can handle both client side and server side validations
- For configuring the validation framework **validators.xml** file or **annotation** is used
- If **validators.xml** file is **not present** in **class path**, the **default.xml** present in com/opensymphony/xwork2/validator/validators/folder is used
- Data of the action class is validated using **validate()** method

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Struts 2 Framework Validator (Cont'd)

Validators Types

Field Validators

- Implemented on **fields accessible** using action
- Field validators are bound to individual fields and can display specific error messages for individual fields

Non Field Validators

- Implemented at **action level**
- Non field validators are bound to action and display action specific messages

Example : Field Validator

Validator.xml

```
<validators>
  <field name="bar">
      <field-validator type="required">
          <message>You must enter a value for bar.</message>
      </field-validator>
  </field>
```

Design Source

Example : Non Field Validator

Validator.xml

```
<validators>
      <validator type="expression">
            <param name="expression">foo lt bar</param>
            <message>Foo must be greater than Bar.</message>
      </validator>
</validators>
```

Design Source

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

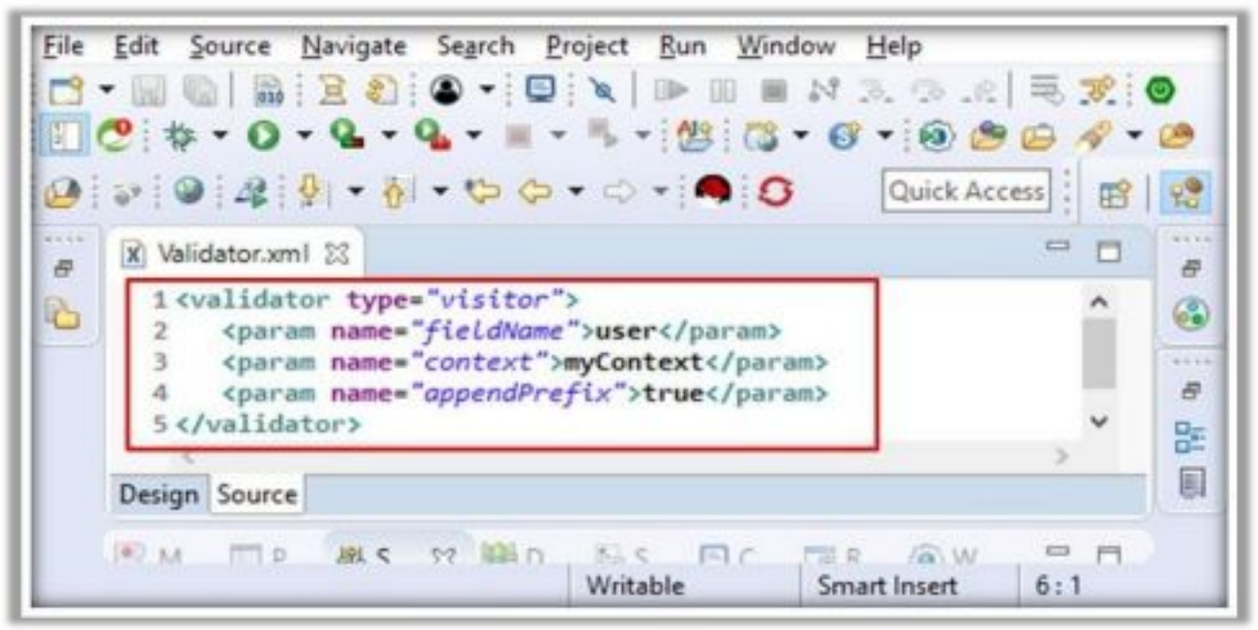
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Struts 2 Framework Validator (Cont'd)

Visitor Field Validators

- Implemented on **model properties** of the action
- Can validate simple object properties, collection of objects, arrays etc.

Example : Visitor Field Validator



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Struts 2 Framework: Built-in Data Validators

List of Some Built-in Data Validators

Validator Name	Class Name	Validation
required	RequiredFieldValidator	If field is not null
requiredstring	RequiredStringValidator	If string not-null and contains length > 0
int	IntRangeFieldValidator	If integer within a certain range
stringlength validator	StringLengthFieldValidator	If String field is of a certain length based on minlenth and maxlength specified
email validator	EmailValidator	If field contains valid email address
date validator	DateRangeFieldValidator	if the date field is within a specified parameter range
conditionalvisitor	ConditionalVisitorFieldValidator	Conditional validator will forward the request to the conditionalfield validator if the expression is true
Expression	ExpressionValidator	Validated based on regular expression supplied
fieldExpression	FieldExpressionValidator	Validates a field based on OGNL expression
Double	DoubleRangeFieldValidator	if double within a certain range
url	URLValidator	If the field value is valid URL
regex	RegexFieldValidator	Validates a field based on regular expression

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Struts 2 Framework Annotation Based Validators

List of Annotation Based Validators

Annotation Name	Validation
@RequiredStringValidator	If the string is not null and length >0
@RequiredFieldValidator	If the string is not null
@EmailValidator	If the field is valid email address
@RegexFieldValidator	If the field matches the provided regular expression
@IntRangeFieldValidator	If the numeric field is within specified range
@StringLengthFieldValidator	If the field length is within the specified minimum and maximum length
@DateRangeFieldValidator	If the date field is within specified range
@FieldExpressionValidator	If the field matches the provided OGNL expression
fieldExpression	FieldExpressionValidator
Double	DoubleRangeFieldValidator
url	URLValidator
regex	RegexFieldValidator

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Struts 2 Custom Validation: Workflow Interceptor

- Implement **Validateable** and **ValidationAware** interface in the action class to create custom validations
- The **validate()** method of Validateable interface should be overridden to implement custom logic
- **Workflow interceptor** is responsible for extracting the defined error details and messages of the action class
- **Field** and **Action** level validations can be implemented by the **ValidationAware** interface

Example: Action class implementing custom validation logic by overriding the validate()

RegisterAction.java

```
1  package Validation;
2  public class RegisterAction extends ActionSupports
3  {
4   private String name,password;
5   public void validate()
6   {
7       if(name.length()<1)
8       {
9           addFieldError("name","Name can't be blank");
10
11      }
12      if(password.length()<1)
13      {
14          addFieldError("password","Password must be greater than 5");
15
16      }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Struts 2 AJAX Validation: jsonValidation Interceptor

- To implement AJAX validation use **jsonValidation** interceptor along with validation interceptor
- jsonValidationWorkflowStack package contains **jsonValidation**, validation and workflow interceptors and basicstack to implement AJAX Validations

Steps to Implement AJAX Validation

- Design input from
- Inherit the ActionSupport class in the action
- Implement validation in validation.xml file
- Define error message
- Register jsonValidationWorkflowStack in Validator.xml

Example: Validator.xml with Result and jsonValidationWorkflowStack Interceptor

Validator.xml

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
3  Configuration 2.1//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">
4 <struts>
5 <package name="a" extends="struts-default">
6 <action name="register" class="mypack.Register">
7 <interceptor-ref name="jsonValidationWorkflowStack"></interceptor-ref>
8 <result name="success">/welcomepage.jsp</result>
9 <result name="input">/indexpage.jsp</result>
10 </action>
11 </package>
12 </struts>
```

Design Source

Note: To implement AJAX validation include dojo plugin in your project

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Data Validation: Spring Framework

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Spring Validator

- **Spring Validator** supports various validation methods, easy to **customize** and **implemented** as plugin to support any validator
- The spring validator interface **org.springframework.validation.Validator** is used to validate objects on server. It contains two methods
  - Support (returns Boolean): Checks if the target class can be validated by the validator
  - Validate : Performs the validation
- In spring the validator interface uses the error object for providing error report of validation
- While validating **error object reports** about failure validation
- Annotations are also used to validate spring application like **@Validator**

Example using Spring Validator

DataValidation.java

```
1  package com.myproject.webapp;
2
3  import org.springframework.validation.Errors;
4  import org.springframework.validation.ValidationUtils;
5  import org.springframework.validation.Validator;
6
7  public class DataValidation implements Validator {
8
9   employee emp=new employee();
10     @Override
11     public void validate(Object target, Errors errors) {
12         ValidationUtils.rejectIfEmpty(errors, "PersonName", "PersonName.empty");
13         employee pemp=(employee)target;
14
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Data Validation: Spring MVC Framework

Example: Spring MVC Data Validation

HomeController.java

```
1  package com.ECouncil.CASE;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.ui.Model;
4  import org.springframework.validation.BindingResult;
5  import org.springframework.web.bind.annotation.ModelAttribute;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RequestMethod;
8
9  @Controller
10 public class HomeController {
11
12
13     private UserValidator uservalidator;
14     @RequestMapping(value = "/UserRegistration", method = RequestMethod
15     public String registration(@ModelAttribute("userform") Userform use
16     {
17         uservalidator.validate(userform, bindresult);
18         return "registration";
19     }
20
```

UserValidator.java

```
1  package com.ECouncil.CASE;
2
3  import org.springframework.validation.Errors;
4  import org.springframework.validation.ValidationUtils;
5  import org.springframework.validation.Validator;
6
7  public class UserValidator implements Validator {
8
9      @Override
10     public boolean supports(Class<?> arg0) {
11         return false;
12     }
13
14     @Override
15     public void validate(Object o, Errors errors)
16     {
17         User myuser = (User) o;
18     ValidationUtils.rejectIfEmptyOrWhitespace(errors, "myusername", "NotEmpty");
19 if(myuser.getUsername().length() < 6 || myuser.getUsername().length() > 32) {
20      errors.rejectValue("myusername", "Size.userform.myusername");
21     } if (userService.findByUsername(myuser.getUsername()) != null) {
22         errors.rejectValue("myusername", "Duplicate.userform.myusername"); }
23
24     }
25
26     }
```

- **Validate()** method of the userValidator class validates userform and binds the result
- The class **UserValidator** implements Validator interface at model class and this class object is used in the controller

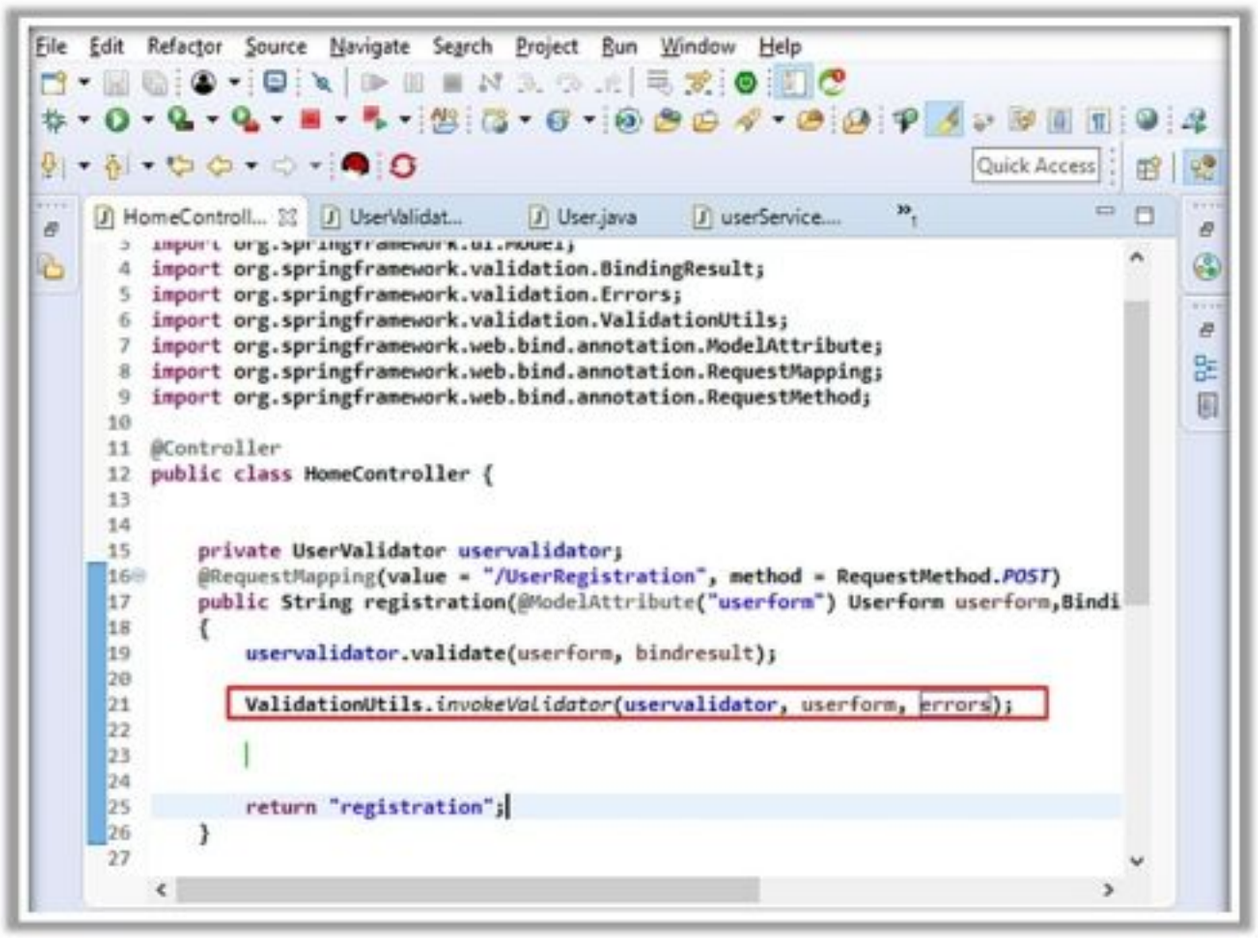
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Implementing Validator

- Validator can be implemented using **invokeValidator()** method in the controller code



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

JSR 380 Bean Validator API

- JSR 380 is the **Java API** specification for **Bean Validation**
- It is used to validate model properties using annotations

- **Standard JSR Annotations**
  - **@NotNull:** It will validate value is not null
  - **@AssertTrue:** It will validate Boolean value
  - **@Size:** This validates the size of the attribute like for string size min or max
  - **@Min:** It validates minimum size value of the attribute
  - **@Max:** It validates maximum size value attribute
  - **@Email:** validates the Email address
  - **@NotEmpty:** It validates Not null property
  - **@NotBlank**
- **Defining Error message globally**

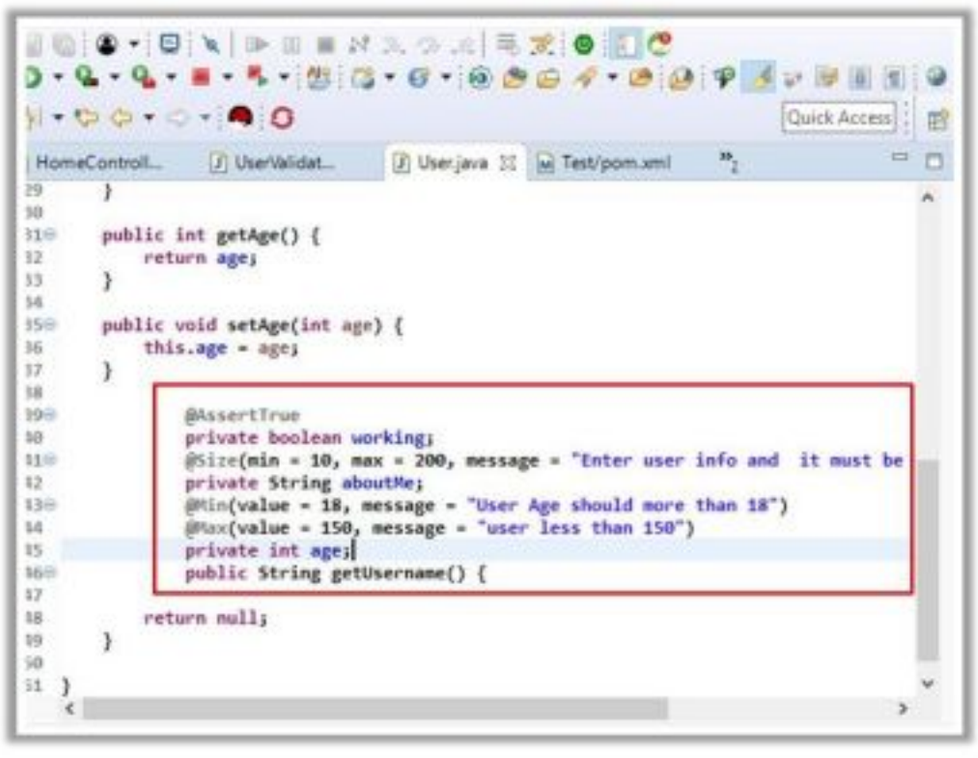
```
javax.validation.constraints.Min.message=Value
must be at least {value}
```

- **Defining Error message as property**

```
@Max(value=120, message="{age.too.old}")
//Annotation

age.too.old=Age must be under 120.// Message
bundle property
```

Manually Passing Error Message



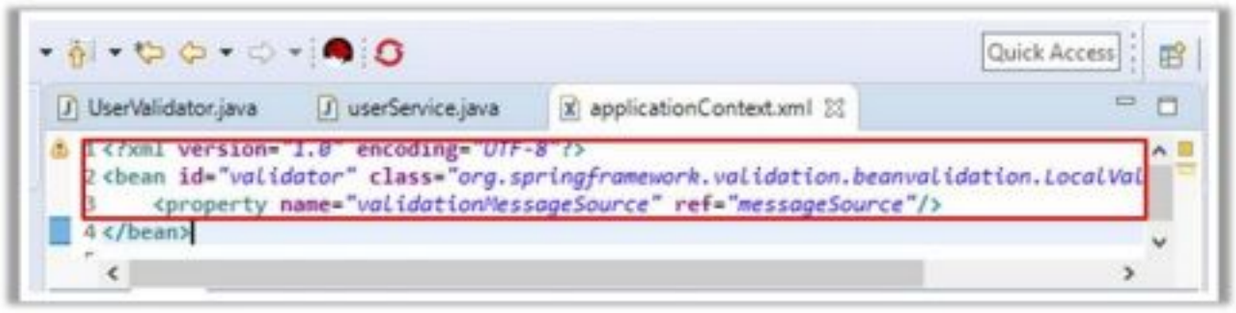
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Configuring JSR 380

Configure validator in the applicationContext



***-servlet.xml**



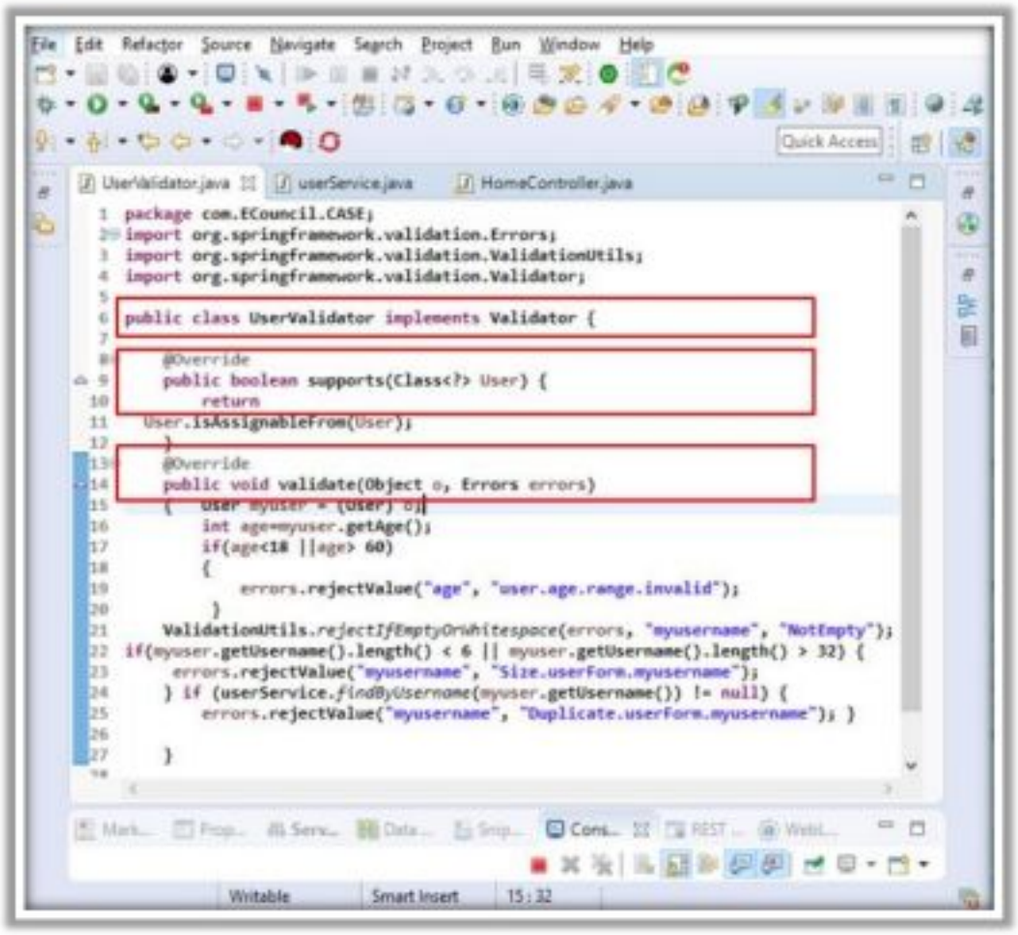
Spring MVC will automatically validates inputs from the controller



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Custom Validator Implementation in Spring

- Custom validation is required for some special business parameter values, for example validating age between 18 to 60
- Custom validator can be implemented using **Validator** interface
- Override the **supports**(Class mycls) method
- Override **validate**(Object o, Errors errors) method



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Spring Validation and Security

1. Spring validation is done to prevent attacks caused through **unchecked input**
2. All the inputs should be checked using the **spring validator** before they are processed by the application
3. Each and every field included in the form should be validated in the **corresponding validation form**
4. Spring support groups and roles in application
5. Vulnerabilities in input validation may lead to **buffer overflow attacks**, **cross-site scripting**, **SQL injection**, etc.

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Input Validation Errors

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Input Validation Errors: Improper Sanitization of Untrusted Data

- Web applications accept data from **untrusted sources**, **contradicted users** and **network connections** and then transfer data from one trusted boundary to another trusted domain to execute
- The data travels across domains in string format with some **internal syntactic structure** that the subsystem should **parse**
- If the subsystem cannot handle the **malicious input** properly it may lead to different types of injection attacks such as SQL injection, XML injection, etc.

Vulnerable Code for SQL injection

- The vulnerable code is an example of SQL injection vulnerability that occurs because the **SQL statement** used in the code **sqlString** accepts unsanitized input arguments

ValidateUser.java

```
3  import java.sql.Connection;
4  import java.sql.ResultSet;
5  import java.sql.SQLException;
6  import java.sql.Statement;
7
8  public class ValidateUser {
9
10     public void doPrivilegedAction(String Uname, char[] pwd)
11 throws SQLException {
12 Connection conn = getConnection();
13 if (conn == null) { // handle error
14 }
15 try {
16 String passwd = hashPasswd(pwd);
17 String sqlString = "SELECT * FROM db_user WHERE username = '"
18 +Uname+ "' AND password = '" + passwd + "'";
19 Statement statmt = conn.createStatement();
20 ResultSet rSet = statmt.executeQuery(sqlString);
21 if (!rSet.next()) {
22 throw new SecurityException("Incorrect User name or password");
23 }
24 }catch (Exception e) {
25     // TODO: handle exception
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Input Validation Errors: Improper Sanitization of Untrusted Data (Cont'd)

Secure Code for SQL Injection

- For sanitizing untrusted input variables, JDBC library is provided with an API for building SQL commands
- The **java.sql.PreparedStatement** class when used properly can escape input strings preventing SQL injection
- The **set*()** method of **preparedstatement** class performs strong type checking of user inputs

- In the following code, **doPriviledgedAction()** method uses **preparedstatement** instead of **java.sql.statment**
- This method also **validates** the **length** of username argument, preventing it from executing an absurdly long username intentionally submitted by an attacker

ValidateUser.java

```
7  public class ValidateUser {
8
9      public void callme(String Uname, char[] pwd) throws SQLException {
10
11
12         Connection conn = getConnection();
13                 if(conn == null) {
14         }
15         try {
16             String passwd = hashPassword(pwd);
17             if (Uname.length() > 8) {
18             }
19             String sqlString ="select * from db_user where Uname=? and pwd=?";
20             PreparedStatement prdstmt = conn.prepareStatement(sqlString);
21             prdstmt.setString(1, Uname);
22             prdstmt.setString(2, passwd);
23             ResultSet rSet = prdstmt.executeQuery();
24             if (!rSet.next()) {
25                 throw new SecurityException("Incorrect username or password");
26             }
27         }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Input Validation Errors: Improper Sanitization of Untrusted Data (Cont'd)

Vulnerable Code for XML Injection

ValidateUser.java

```
1  package Validation;
2
3  import java.io.BufferedOutputStream;
4  import java.io.IOException;
5
6  public class ValidateUser {
7
8      private void createXMLStream(BufferedOutputStream bufOpStrm, String quantity) throws
9
10         String xmlStr = "<item>\n<description>Widget</description>\n" + "" + "<price>500
11                 + quantity + "</quantity></item>";
12         bufOpStrm.write(xmlStr.getBytes());
13         bufOpStrm.flush();
14     }
15
```

- In the code, for sending an XML query to a server, a simple **string concatenation** is used by a client method
- The client method used however **does not perform input validation** properly leading to an XML injection

Secure Code for XML Injection

```
5  import java.util.regex.Pattern;
6
7  public class ValidateUser {
8
9      private void createXMLStream(BufferedOutputStream bufOpStrm, String quantity) throws
10
11         if(!Pattern.matches("[0-9]+", quantity)) {
12
13         }
14         String xmlStr = "<item>\n<description>Widget</description>\n" +""
15                 + "<price>500.00</price>\n" +"<quantity>"+quantity+"</quantity></item>";
16         bufOpStrm.write(xmlStr.getBytes());
17         bufOpStrm.flush();
18         }
```

- In the code snippet **whitelisting method** is used to sanitize inputs
- The method **strictly validates** quantity field input numbers between 0 and 9

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Input Validation Errors: Improper Validation of Strings

- Web applications that accept input strings from untrusted sources perform **filtering** and **validation mechanisms** based on the strings' character data
- Unicode standards are followed for character information in Java by:
  - Checking if two strings are **equivalent** to each other
  - Transforming a particular unicode normalization form to either **canonical** or **compatibility** equivalence

Normalization Forms	
Form	**Description**
Normalization Form D (NFD)	Canonical Decomposition
Normalization Form C (NFC)	Canonical Decomposition, followed by Canonical Composition
Normalization Form KD (NFKD)	Compatibility Decomposition
Normalization Form KC (NFKC)	Compatibility Decomposition, followed by Canonical Composition

- Using normalization forms KC and KD for arbitrary input strings may sometimes remove **formatting distinctions** that are important for text semantics
- NFKC converts the input strings into an equivalent **canonical form** without altering formatting distinctions to the required input form
- The i**Normalize method** s used to convert **Unicode text** into an equivalent **composed** or **decomposed form** making sorting and searching of text easier

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Input Validation Errors: Improper Validation of Strings (Cont'd)

- In the below code, String is **validated before normalizing** and it fails to detect any arbitrary inputs
- Validation logic also fails to detect inputs as a check for angle brackets does not detect alternate **Unicode representations**

- In the below code, validating is performed after normalizing string into **canonical angle brackets**
- Input validation mechanism throws an **IllegalStateException** if it detects any malicious inputs

Vulnerable Code

ValidateUser.java

```
16
17      String Str = "\uFE64" + "script" + "\uFE65";
18      Pattern ptrn = Pattern.compile("[<>]");
19      Matcher mtchr = ptrn.matcher(Str);
20      if (mtchr.find()) {
21          throw new IllegalStateException();
22      } else {
23      }
24      Str = Normalizer.normalize(Str, Form.NFKC);
25
26
```

Secure Code

ValidateUser.java

```
16
17      String Str = "\uFE64" + "script" + "\uFE65";
18      Str = Normalizer.normalize(Str, Form.NFKC);
19      Pattern ptrn = Pattern.compile("[<>]");
20      Matcher mtchr = ptrn.matcher(Str);
21      if (mtchr.find()) {
22          throw new IllegalStateException();
23      } else {
24      }
25
26
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Input Validation Errors: Improper Logging of User Inputs

- Log injection attacks may occur if a log contains **malicious user inputs** due to improper validation mechanism or improper sanitization
- Attackers can split a log entry into two log entries using **carriage return** and **line feed** (CRLF) **sequences** or insert a fake log entry
- Also, logging malicious inputs that are not sanitized can result in **information leakage**

Vulnerable Code

ValidateUser.java

```
22
23      if (loginCorrect) {
24
25          logger.severe("User login succeeded for: " + username);
26      } else {
27          logger.severe("User login failed for: " + username);
28      }
29
```

- In the code, username is logged for **invalid user request** without proper validation

Secure Code

```
    if((!Pattern.matches("[A-Za-z0-9_]+", username)))
    {
        logger.severe("User login failed for unauthorized user");
    }
    else if (loginCorrect) {

    logger.severe("User login succeeded for: " + username);

    }
    else {
    logger.severe("User login failed for: " + username);

    }
31
32
```

- In the code, validation is performed before **logging username**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Input Validation Errors: Improper Incorporation of Malicious Inputs into Format Strings

- In Java, **format strings** implementation is **stricter** than in any other languages
- If any conversion argument **fails** to **match** with the corresponding **format specifier**, implementation method throws an **exception**
- This method aims at **limiting malicious exploits**; although a malicious user input can yet exploit format strings and cause denial of service or data leaks
- It is advisable to not **incorporate** any **input strings** of untrusted sources into format strings

Vulnerable Code

ValidateUser.java

```
1  package Validation;
2
3  import java.util.Calendar;
4  import java.util.GregorianCalendar;
5  public class ValidateUser {
6      static Calendar cal=new GregorianCalendar(1995, GregorianCalendar.MAY, 23);
7          public static void main(String[] args)
8          {
9              // args[0] is the credit card expiration date
10             // args[0] can contain either %1$tm, %1$te or %1$tY as malicious arguments
11             // First argument prints 05 (May), second prints 23 (day)   and third prints
12             // Perform comparison with cal, if it doesn't match print the following line
13 System.out.printf(args[0] +
14         " did not match! HINT: It was issued on %1$terd of some month", cal);
15         }
```

- The code describes an **information leak** issue
- It accepts the expiration date of a credit card and incorporates it within the format string

Secure Code

```
dar;
gorianCalendar(1995, GregorianCalendar.MAY, 23);
        public static void main(String[] args)
        {

            // args[0] is the credit card expiration date
            // Perform comparison with cal, if it doesn't match print the following line
        System.out.printf("%s did not match! "+ " HINT: It was issued on %1$terd of some month", args[0],cal);
    }
}
```

- In the code, it is ensured that user inputs are not **incorporated** in the **format string**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Input Validation Errors: Inappropriate Use of Split Characters in Data Structures

- The Java language string occupies **16 bits**
- Java char cannot represent various possible **Unicode characters**
- Splitting a char or byte array risks splitting a **multi byte character**
- Attackers can bypass the input validation checks when characters are split between two data structures

Vulnerable Code

ValidateUser.java

```
1  package Validation;
2
3  import java.io.IOException;
4  import java.io.InputStream;
5  import java.net.Socket;
6
7  public class ValidateUser {
8
9      public final int MaxSize = 1024;
10     public String read_Bytes(Socket skt) throws IOException {
11         InputStream ins = skt.getInputStream();
12         byte[] data = new byte[MaxSize+1];
13         int offset = 0;
14         int bytesRead = 0;
15         String str = new String();
16         while ((bytesRead = ins.read(data, offset, data.length - offset)) != -1) {
17         offset += bytesRead;
18         str += new String(data, offset, data.length - offset, "UTF-8");
19          if (offset >= data.length) {
20          throw new IOException("Too much input");
21             }
22           }
23          ins.close();
24          return str;
25         }
26 }
```

- The code fails to interact between characters of **multibyte encoding** and **boundaries** between the **loop iterations**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Input Validation Errors: Inappropriate Use of Split Characters in Data Structures (Cont'd)

- The code eliminates the splitting of **multi byte encoded characters** in the buffers
- It also defers the construction of the **result string** until the data is read completely

Secure Code

ValidateUser.java

```
2
3  import java.io.IOException;
4  import java.io.InputStream;
5  import java.net.Socket;
6
7  public class ValidateUser {
8
9      public final int MaxSize = 1024;
10     public String read_Bytes(Socket skt) throws IOException {
11         InputStream ins = skt.getInputStream();
12         byte[] data = new byte[MaxSize+1];
13         int offset = 0;
14         int bytesRead = 0;
15         while ((bytesRead = ins.read(data, offset, data.length - offset)) != -1) {
16         offset += bytesRead;
17         if (offset >= data.length) {
18         throw new IOException("Too much input");
19         }
20         }
21         String str = new String(data, "UTF-8");
22         ins.close();
23         return str;
24
25         }
26 }
27
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Input Validation Errors: Improper Validation of Non-character Code Points

- Attackers can bypass the validation checks if the input validation is done without the removal of **non-character code points**

Vulnerable Code

- The code below shows how only valid ASCII characters are accepted
- It also shows deletion of **non-ASCII characters validation** without deleting non-ASCII characters
- It allows an attacker to bypass the validation checks by changing the **<script> tag**

validateUser.java

```
package Validation;

import java.io.BufferedOutputStream;
import java.io.IOException;
import java.text.Normalizer;
import java.text.Normalizer.Form;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class ValidateUser {

    private void createXMLStream(BufferedOutputStream bufOpStrm, String quantity) throws IOException {

        String str = "<scr" + "\uFEFF" + "ipt>";
        str = Normalizer.normalize(str,Form.NFKC);
                Pattern ptrn = Pattern.compile("<script>");
        Matcher mtchr = ptrn.matcher(str);
        if (mtchr.find()) {
          System.out.println("Found black listed tag");
        } else {

        }
        str = str.replaceAll("^\\p{ASCII}]","");
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Input Validation Errors: Improper Validation of Non-character Code Points (Cont'd)

Secure Code

- The code below gives example of how a **Unicode sequence \uFFD** is replaced with some unknown characters
- <script> tag is checked before **performing sanitation**
- This process avoids malicious input from **bypassing** the **filters**

ValidateUser.java

```
4  import java.io.IOException;
5  import java.text.Normalizer;
6  import java.text.Normalizer.Form;
7  import java.util.regex.Matcher;
8  import java.util.regex.Pattern;
9
10 public class ValidateUser {
11
12   private void createXMLStream(BufferedOutputStream bufOpStrm, String quantity)
13         throws IOException {
14
15     String str = "<scr" + "\uFEFF" + "ipt>";
16     str = str.replaceAll("^\\p{ASCII}]", "\uFFFD");
17     str = Normalizer.normalize(str,Form.NFKC);
18           Pattern ptrn = Pattern.compile("<script>");
19     Matcher mtchr = ptrn.matcher(str);
20     if (mtchr.find()) {
21       System.out.println("Found black listed tag");
22     } else {
23
24     }
25     str = str.replaceAll("^\\p{ASCII}]","");
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Input Validation Errors: Improper Use of String Modification

Vulnerable Code for XML Injection

- String should not be modified after validation. By doing that it allows to attacker to bypass validation
- With the help of Unicode perform the modification of the string to **validate** properly

ValidateUser.java

```
1  package Validation;
2
3  import java.io.BufferedOutputStream;
4  import java.io.IOException;
5
6  public class ValidateUser {
7
8      private void createXMLStream(BufferedOutputStream bufOpStrm, String quantity) throws IOException {
9          String xmlStr;
10         xmlStr = "<item>\n<description>Widget</description>\n" +
11         "<price>500.0</price>\n" +
12         "<quantity>" + quantity + "</quantity></item>";
13         bufOpStrm.write(xmlStr.getBytes());
14         bufOpStrm.flush();
15     }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Input Validation Errors: Improper Use of String Modification (Cont'd)

Secure Code for XML Injection

- In the code snippet the **whitelisting method** is used to sanitize inputs
- The method **strictly validates** quantity field input numbers between 0 and 9

ValidateUser.java

```
1  package Validation;
2
3  import java.io.BufferedOutputStream;
4  import java.io.IOException;
5  import java.util.regex.Pattern;
6
7  public class ValidateUser {
8
9      private void createXMLStream(BufferedOutputStream bufOpStrm, String quantity) throws IOException {
10
11         if(!Pattern.matches("[0-9]+", quantity)) {
12
13         }
14         String xmlStr = "<item>\n<description>Widget</description>\n" +"<price>500</price>\n" +"<quantity>"+quantity+"</quantity></item>";
15         bufOpStrm.write(xmlStr.getBytes());
16         bufOpStrm.flush();
17         }
18
19
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Input Validation Errors: Improper Comparison of Locale-dependent Data

Vulnerable Code for Improper Date Comparison

ValidateUser.java

```
1  package Validation;
2  import java.util.Date;
3  import java.text.DateFormat;
4
5  public class ValidateUser {
6
7      public static void isJune(Date date) {
8          String myString = DateFormat.getDateInstance().format(date);
9          System.out.println("The date is "+myString);
10         if(myString.startsWith("Jun ")) {
11         System.out.println("Its Cool season!");
12         }
13         else {
14             System.out.println("Its not Cool Season");
15         }
16
17     }
18 }
```

- Ensure the date specification as per correct locales
- Improper comparison may lead to unexpected results and bypass of input validations

Secure Code for Locale Comparison

- The code snippet implements US Local while defining the date parameters

```
4  import java.text.DateFormat;
5
6  public class ValidateUser {
7
8      public static void isJune(Date date) {
9          Date rightNow = null;
10         @SuppressWarnings("null")
11         String myString = DateFormat.getDateInstance(DateFormat.MEDIUM, Locale.US).format(rightNow.getTime());
12         System.out.println("The date is "+myString);
13         if(myString.startsWith("Jun ")) {
14         System.out.println("Its Cool season!");
15         }
16         else {
17             System.out.println("Its not Cool Season");
18         }
19
20     }
21 }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

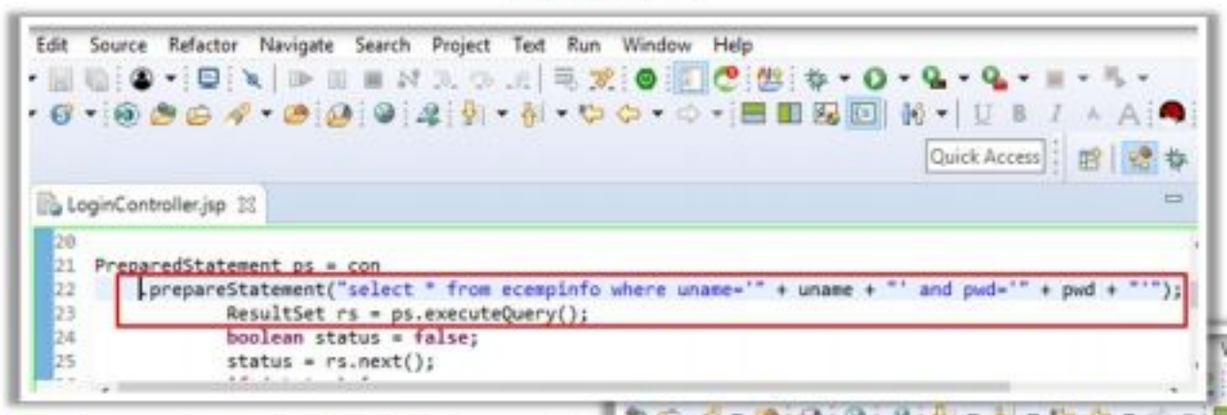
Common Secure Coding Practices

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection

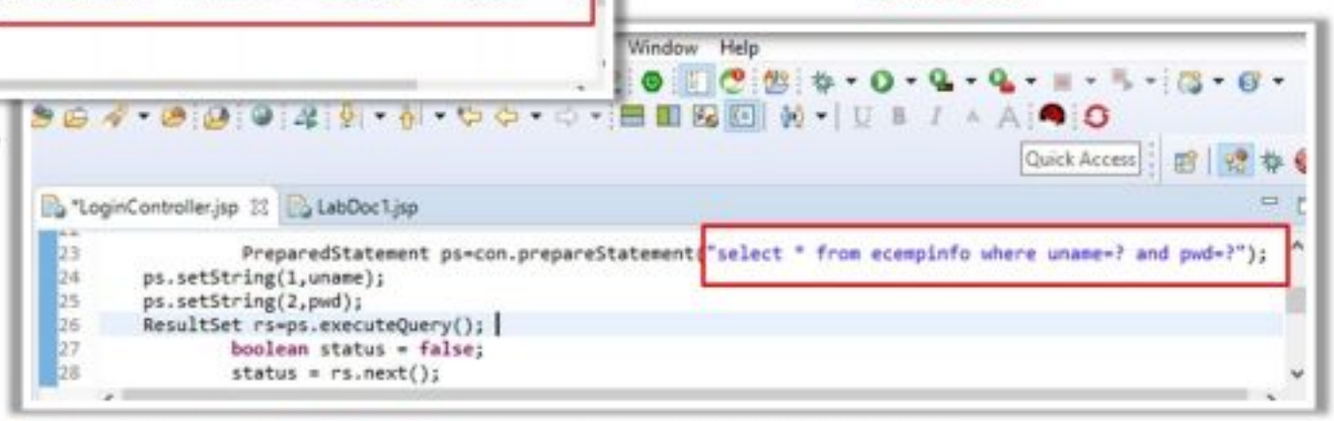
- SQL injection is a code injection technique that uses the **security vulnerability** of a database for its attacks
- Attackers inject a **malicious code** into the strings that are later on passed on to SQL server for execution

Insecure Code



- In the code, **concatenating strings** are used to form an SQL query resulting in SQL injection attacks

Secure Code



- In the code, **parameterized queries** are used to prevent SQL injection attacks

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

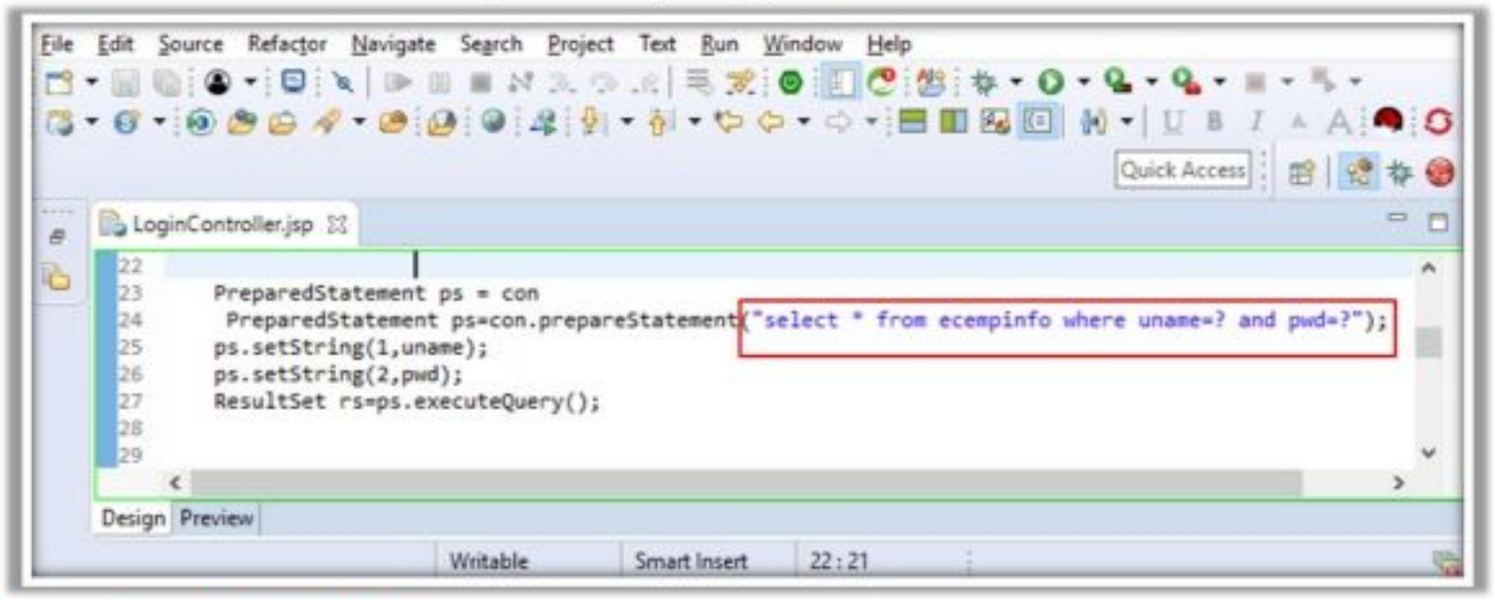
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Prepared Statement

- Prepared statements define **methods** and **properties** that are used to send SQL commands to retrieve data from the database
- They can be used with or without **parameters**
- Increased performance of the code in case of repeated SQL statements
- They are also known as **pre-compiled statements**
- Prepared statements are supported by various DBMS like **Oracle**, **MySQL**, **DB2**, **SQL Server** and **PostgreSQL**

Example: Creating a Prepared Statement

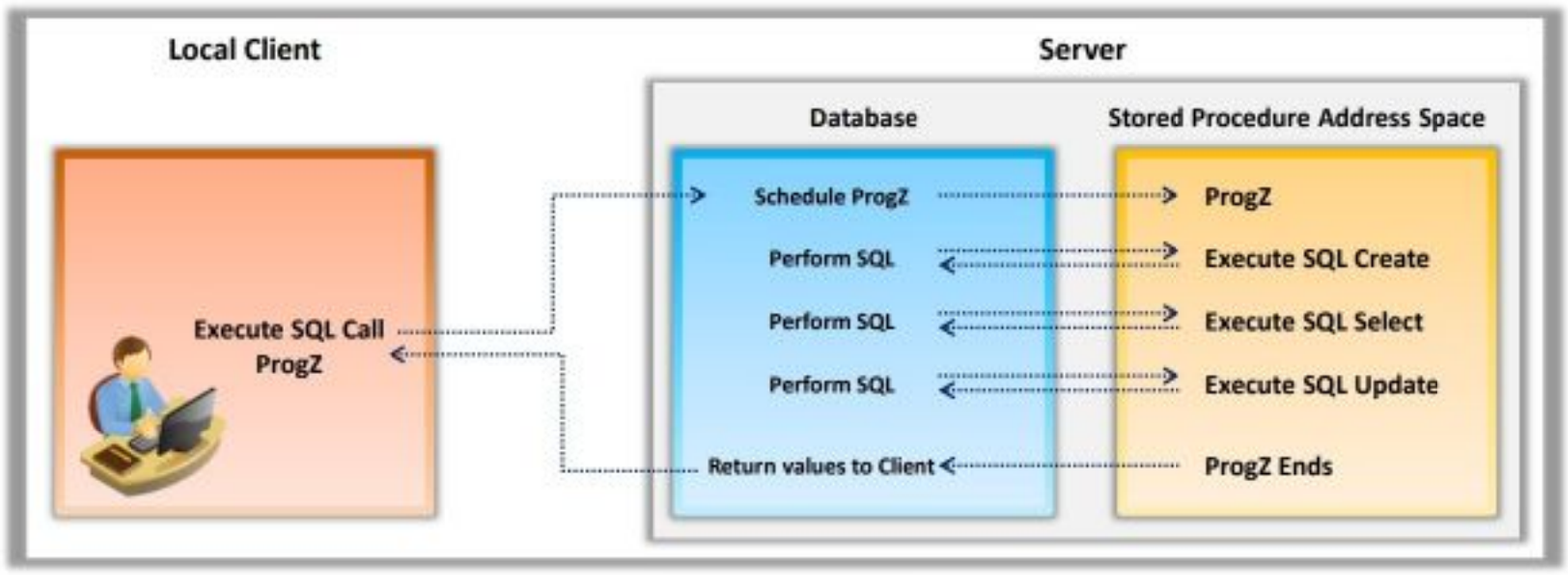


- The '**?**' is the **parameter marker** that represents the parameters (in JDBC)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Stored Procedures

- Stored procedures are user written programs that are stored in **database servers** and can be invoked by client applications
- Stored procedures can **execute SQL statements** at the server, as well as application logic level, for a specific function
- The SQL CALL statement is part of the **SQL ISO/ANSI standard** for invoking stored procedures from database management systems
- Client applications can invoke a stored procedure by using a single **SQL CALL statement**
- Client programs can **pass** and **receive** parameters from stored procedure



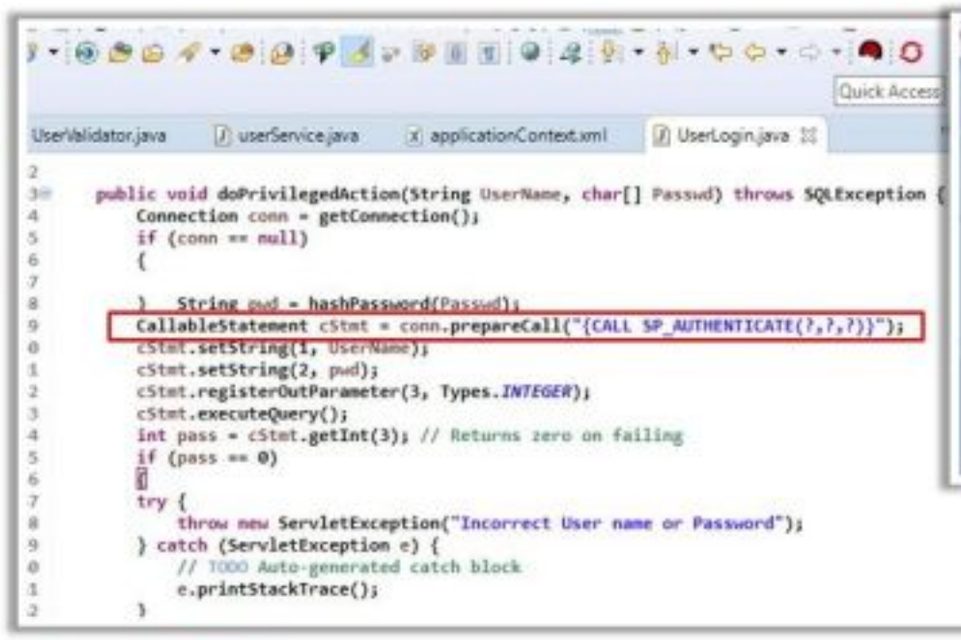
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

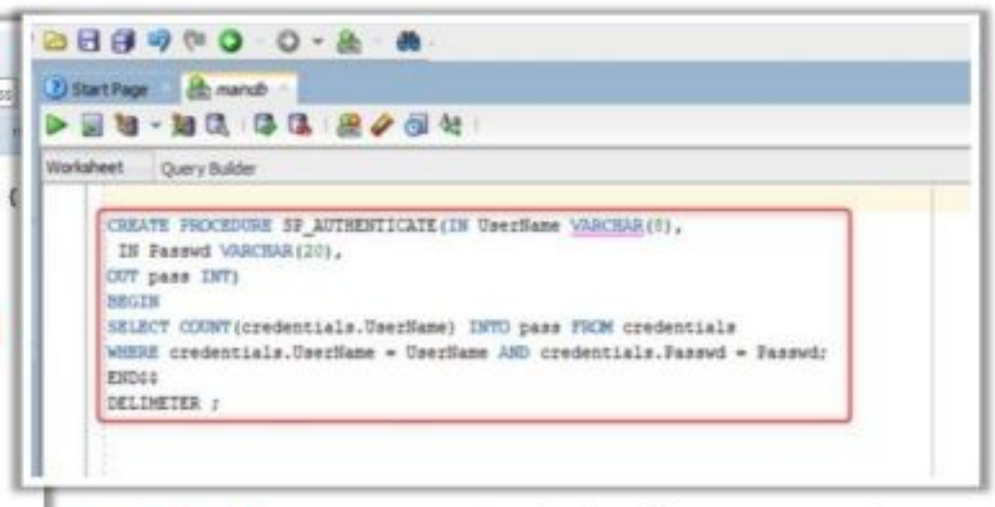
Vulnerable and Secure Code for Stored Procedures

Example: Implementing Stored Procedure



```
2
3   public void doPrivilegedAction(String UserName, char[] Passwd) throws SQLException {
4       Connection conn = getConnection();
5       if (conn == null)
6       {
7
8       }   String pwd = hashPassword(Passwd);
9       CallableStatement cStmt = conn.prepareCall("{CALL SP_AUTHENTICATE(?,?,?)}");
0       cStmt.setString(1, UserName);
1       cStmt.setString(2, pwd);
2       cStmt.registerOutParameter(3, Types.INTEGER);
3       cStmt.executeQuery();
4       int pass = cStmt.getInt(3); // Returns zero on failing
5       if (pass == 0)
6       {
7       try {
8           throw new ServletException("Incorrect User name or Password");
9       } catch (ServletException e) {
0           // TODO Auto-generated catch block
1           e.printStackTrace();
2       }
```

- **java.sql.CallableStatement** API is used to execute SQL Stored Procedures. **CallableStatement** is not vulnerable to SQL Injection unless the called stored procedure code is vulnerable.



```
CREATE PROCEDURE SP_AUTHENTICATE(IN UserName VARCHAR(8),
 IN Passwd VARCHAR(20),
OUT pass INT)
BEGIN
SELECT COUNT(credentials.UserName) INTO pass FROM credentials
WHERE credentials.UserName = UserName AND credentials.Passwd = Passwd;
END$$
DELIMETER ;
```

- The **SELECT** statement restricts the size of parameters and length of input variables

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Stored **Procedures** for Securing Input Validation

1. Stored procedures enable a user to **split** the **application logic** between the client and server
2. This technique is used to prevent the client applications from **manipulating** the **contents** of the server data
3. Stored procedures are secured through **encoding** in the **database** and require admin privileges to execute the program
4. Stored procedures perform input validation of data by **verifying text**, **characters** and **numeric data**
5. Stored procedures prevent SQL injection attacks by performing thorough validation on any user input before **processing** the **data**
6. Stored procedures use a safe interface such as a **callable statement** in JDBC (Java Database Connectivity)
7. Stored procedures also manage **network traffic** where a single SQL statement can be used to send and receive a series of SQL statements

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

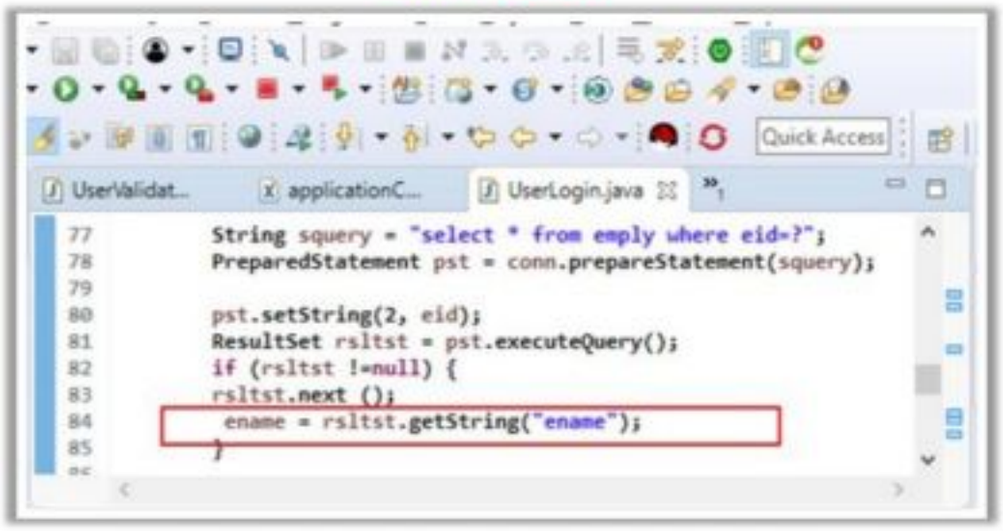
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Cross-site Scripting (XSS)

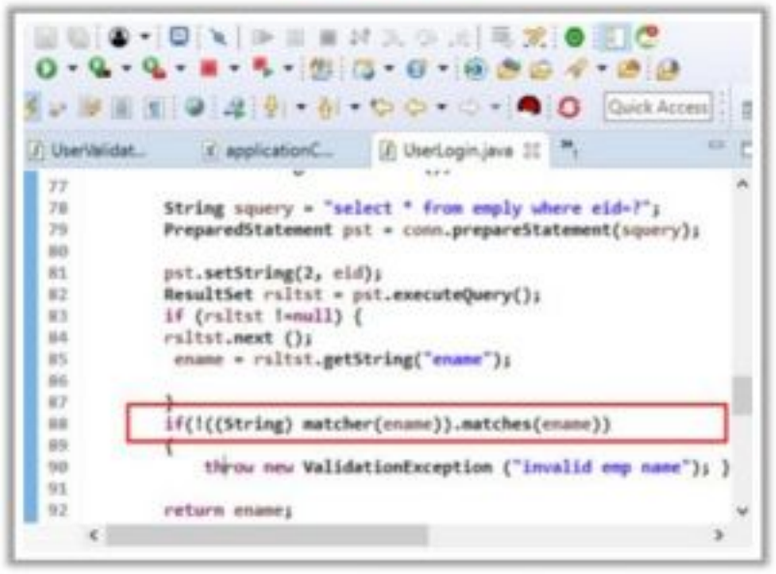
- Cross-site scripting ('XSS' or 'CSS') attacks exploit vulnerabilities in **dynamic environment web pages** that enable attackers to inject client-side script into web pages viewed by other users

Insecure Code



- The insecure code snippet queries a database for an employee id **without proper input validation mechanism** and prints the employee name

Secure Code



- The secure code snippet **uses input validation** and **output encoding** to prevent attackers from executing any malicious scripts

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Whitelisting vs Blacklisting

Whitelisting

- Whitelisting is a process to **verify** the **inputs** against possible expected known values
- The **maintenance costs** of blacklist coded applications are more than the whitelist coded one
- As a whitelist expects only **known good desired values** and rejects every thing else, therefore it is a good approach in input validation
- **Regular expressions** help in creating a whitelist for the **secured code**
- Before **implementing** regular expressions in code, **test** the regular expressions itself

Blacklisting

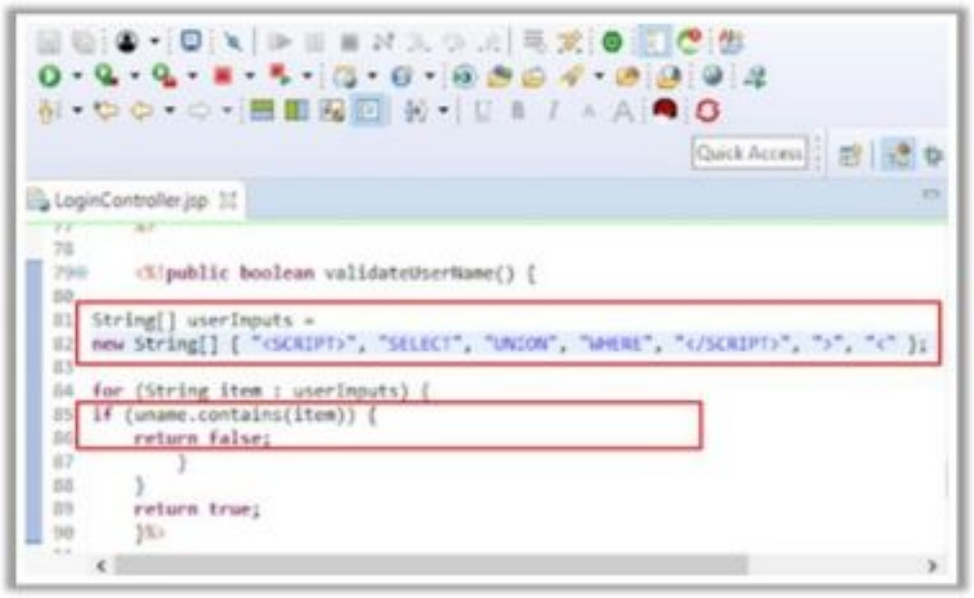
- Testing input values against a set of negative expected inputs is called **blacklisting**
- Checking all the input values received with that of the known **bad** or **negative conditions** is called blacklisting
- Developers cannot think of every possible bad input **permutation** and insert it in the code
- Therefore a blacklist works well only with the list created by the developer, but all new **vectors** will be missed

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

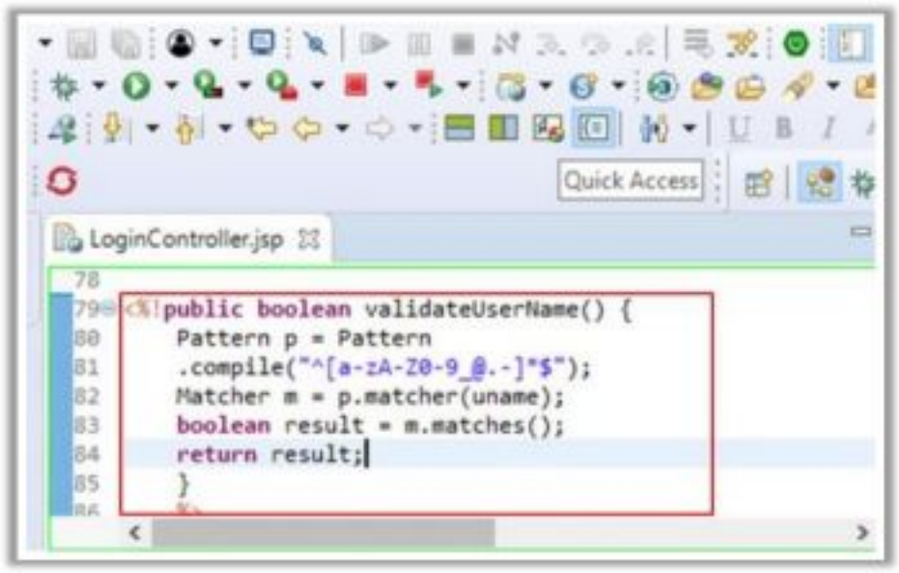
Vulnerable and Secure Code for Blacklisting & Whitelisting

Vulnerable Code



- In the code, the **validateUserName** function uses the blacklisting approach

Secure Code



- In the code , the **validateUserName** function uses the whitelisting approach

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Vulnerable and Secure Code for Blacklisting & Whitelisting (Cont'd)

Vulnerable Code

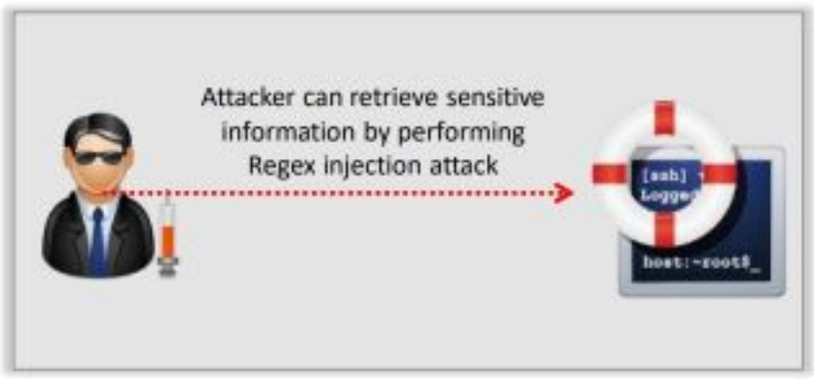


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Regular Expressions

- A regular expression provides a **concise** and **flexible** means to "match" (specify and recognize) strings of text, such as particular **characters**, **words**, or **patterns** of characters
- There are even chances of modifying the input of a regular expression by the attacker; such type of an attack is known as a **regex injection attack**
- Proper input validation helps in avoiding such attacks



Expression	Description
(013)	A single digit 0, 1, or 3
(0-9) (0-9)	Any two-digit number from 00-99
(0-9 && (^4 5 6 7)	A single digit that is 0, 1, 2, 3, 8, or 9
(a-z 0-9)	A single character that is either a lowercase letter or a digit
(a-z A-z) (a-z A-Z 0-9_$)	A valid Java identifier consisting of alphanumeric characters, underscores and dollar signs with the first character being an alphabet
(wb) (ad I eed)	Matches wad, weed, bad, and beed
(AZ I CA I CO) (0-9) (0-9)	Matches AZxx, CAxx and COxx where x is a single digit

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Vulnerable and Secure Code for Regular Expressions

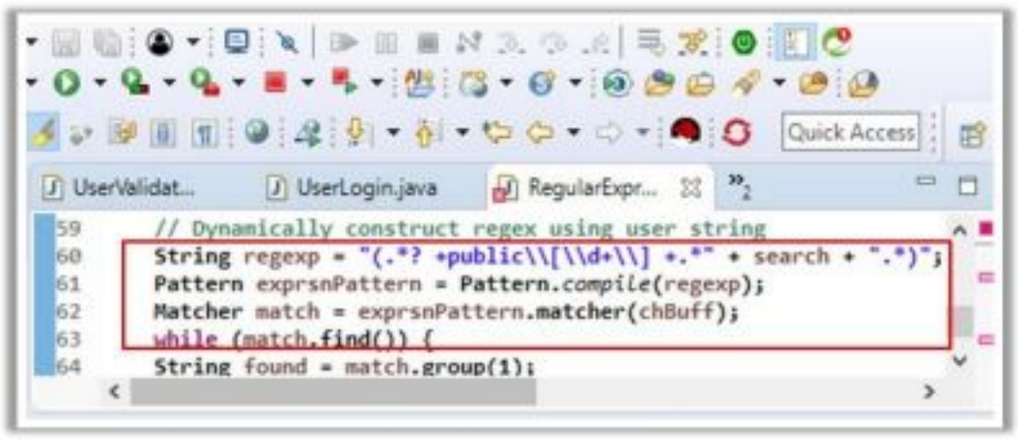
Vulnerable Code

UserValidator.java | userService.java | applicationContext.xml | UserLogin.java

```
24          // Log file is mapped into the memory and reloaded
25
26
27
28  try {
29      FileChannel fchannel = new FileInputStream(
30
31      "path").getChannel();
32          //Getting size of the file and mapping it
33              int fsize = (int) fchannel.size();
34          final MappedByteBuffer mapBuffer = fchannel.map(
35          FileChannel.MapMode.READ_ONLY, 0, fsize);
36          Charset cs = Charset.forName("ISO-8859-15");
37          final CharsetDecoder csDecoder = cs.newDecoder();
38      chBuff = csDecoder.decode(mapBuffer); // File is read into char buffer
39      Runnable periodicLogRead = new Runnable() {
40      @Override public void run() {
41      synchronized(lock) {
42
43
44      }
45      }
46      };
47      scheduler.scheduleAtFixedRate(periodicLogRead,0, 5, TimeUnit.SECONDS);
48
49  }catch(Exception e)
50  {
51      }
52  }
53
54
55      public Set<String> suggestSearches(String search) {
56      CharBuffer chBuff = null;
57      synchronized(lock) {
58      Set<String> str_srch = new HashSet<String>();
59      // Dynamically construct regex using user string
60      String regexp = "(.*? +public\\[\\d+\\] +.*" + search + ".*)";
61      Pattern exprsnPattern = Pattern.compile(regexp);
62      Matcher match = exprsnPattern.matcher(chBuff);
63      while (match.find()) {
64      String found = match.group(1);
65      str_srch.add(found);
66      }
67      return str_srch;
```

- The sample code regularly **loads** a **log file** into **memory** and clients are allowed to get the keyword search suggestions after passing the **keyword** as an argument to **suggestSearches()**
- This code might also lead to Regex injection attacks

Secure Code



- This secure code filters the **non-alphanumeric characters** excluding '**space**' and '**single quote**' from the search string. This code prevents regexp injection

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Character Encoding

- In character encoding systems, code is paired with each character from a presumed **repertoire** that may include bit patterns, octets and number sequences
- **American Standard Code for Information Interchange** (ASCII) and **Unicode** are the recognized standards used in character encoding systems
- Character encoding is related to input/output where text is transmitted between different systems using **encoding schemes**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Vulnerable and Secure Code for Character Encoding

Vulnerable Code

CharacterEncoding.java

```
1  import java.io.DataInputStream;
2  import java.io.FileInputStream;
3  import java.io.IOException;
4
5  public class CharacterEncoding {
6
7
8      public static void main(String args[]) {
9
10         FileInputStream fInStr = null;
11         try {
12         fInStr = new FileInputStream("AnyFile");
13         DataInputStream dInStr = new DataInputStream(fInStr);
14         byte[] bData = new byte[1024];
15
16         dInStr.readFully(bData);
17         String result = new String(bData);
18
```

- In the code, the difference between **default encoding** and **byte array** produced during encoding causes **undefined behavior** for inputs

Secure Code

*CharacterEncoding.java

```
2  import java.io.FileInputStream;
3  import java.io.IOException;
4
5  public class CharacterEncoding {
6
7
8      public static void main(String args[]) {
9
10         FileInputStream fInStr = null;
11         try {
12         fInStr = new FileInputStream("AnyFile");
13         DataInputStream dInStr = new DataInputStream(fInStr);
14         byte[] bData = new byte[1024];
15
16         dInStr.readFully(bData);
17         String encoding="UTF-16LE";
18         String result = new String(bData,encoding);
19
```

Writable | Smart Insert | 19 : 1

- The code explicitly specifies intended **character encoding** in the **String constructor**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Checklist for Character Encoding

- Strictly implement standard **ASCII** or **Unicode character** encoding techniques to prevent **injection attacks**
- Sanitize rejected data and don't store erroneous log data in applications
- Create rules for maximum length and usage of special characters for inputs from external sources
- Use **HTML encoding** and **URL encoding** for user inputs to restrict script injection attacks from running in the browser
- Use compatible encodings on both sides of a communication channel to eliminate risks of data corruption
- Create **rules for data validation** by listing input characters from strongest to weakest such as:
  1. Limit exact matches
  2. Accept desirable known
  3. Reject known undesirables
  4. Sanitize known undesirable matches

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Cross-site Scripting (XSS) Countermeasures

1. Use HTML and URL encoding to ensure all the **input values** are validated before being echoed back to the user
2. Take necessary precautions while using **dynamic environments** that execute script and retrieve data
3. Configure the **web browser** to disable scripting
4. Use **filter techniques** that store and process input variables on the server
5. Use properly designed **error handling mechanisms** for reporting input errors
6. Implement **character encoding techniques** for web pages such as ISO-8859-1 or UTF 8

result.jsp

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.
```

Design | Preview

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

HTML Encoding

I. HTML encoding is used to **protect unsafe output** from attackers

II. The variables found in an application should be **validated** and **encoded** before the output is sent to the HTML pages

III. Converting JavaScript and HTML tags through encoding protects the code from **cross-site scripting vulnerability**

IV. User data that does not implement HTML encoding may result in **markup injection**

V. HTML encoding is done by converting the html entities into encodings as `"<script>"`, and is displayed as `"<script>"`

Examples:

- `/ --> /`
- `& --> &`
- `' --> '`
- `< --> <`
- `" --> "`
- `> --> >`

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Vulnerable and Secure Code for HTML Encoding

Vulnerable Code

result.jsp

```
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8855
7 <title>Sample Captcha</title>
8 </head>
9 <body>
10 <%
11 String result=request.getParameter("result");
12 %>
13
14 Student Result:&nbs;<%=result%>
15 </body>
16 </html>
```

Design | Preview

- The given code **requests** input from the user
- The code is **executed correctly** when the user gives valid input
- As there is no input validation, the code becomes **vulnerable** if the user enters **invalid input**

Secure Code

result.jsp

```
19
20
21 <%! public static String HTMLEncode(String TagFragment){
22 final StringBuffer res = new StringBuffer();
23 final StringCharacterIterator itr = new StringCharacterIterator(TagFragment);
24     char chrctr =  itr.current();
25 while (chrctr != StringCharacterIterator.DONE )
26     {
27       if (chrctr == '<')
28            res.append("&lt;");
29       else if (chrctr == '>')
30            res.append("&gt;");
31       else if (chrctr == '\"')
32            res.append("&quot;");
33       else if (chrctr == '\'')
34            res.append("&#039;");
35       else if (chrctr == '\\')
36            res.append("&#092;");
37       else if (chrctr == '&')
38            res.append("&amp;");
39       else {
40 //the char is not a special one
41 //add it to the result as is
42            res.append(chrctr);
43       }
44       chrctr = itr.next();
45     }
46     return res.toString();
```

Design | Preview

- The given code ensures **HTML entities** are encoded
- Each character is **passed dynamically** to the encoding function

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

HTML Encoding using ESAPI Encoder

Vulnerable Code

result.jsp

```
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7 <title>Sample Captcha</title>
8 </head>
9 <body>
10 <%
11 String result=request.getParameter("result");
12 %>
13
14 Student Result:&nbs;<%=result%>
15 </body>
16 </html>
```

Design | Preview

- The given code **requests** input from the user
- The code is **executed correctly** when the user gives valid input
- As there is no input validation, the code becomes **vulnerable** if the user enters **invalid input**

Secure Code

result.jsp

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1"%>
3 <%@ page import="java.text.*" %>
4 <%@page import="org.owasp.esapi.ESAPI" %>
5
6 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
7 <html>
8 <head>
9 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
0 <title>Sample Captcha</title>
1 </head>
2 <body>
3 <%
4 String result=ESAPI.encoder().encodeForHTML(request.getParameter("result"));
5 %>
6
7 Student Result:&nbs;<%=result%>
8
```

Preview

- The given code implements **ESAPI.Encoder** encode for HTML method to encode data to display as HTML

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Cross-site Request Forgery (CSRF)

- Cross-site Request Forgery (CSRF) attacks **exploit web application vulnerabilities** where an attacker sends malicious requests to third party sites through an unsuspecting user's browser
- Usually, vulnerabilities in web application (**GET**/**POST**/**Path parameters**) confront this type of an attack, not the victim's browser or CRPF hosting site

Insecure Code

DefaultServlet.java

```
29
30  protected void doGet(HttpServletRequest request, HttpServletResponse response)
31          throws ServletException, IOException {
32
33      doPost(request,response);
34  }
```

- In the code, attackers can **manipulate** the **GET request** to view any unauthorized data

Secure Code

```
30
31  protected void doGet(HttpServletRequest request, HttpServletResponse response)
32          throws ServletException, IOException {
33
34      response.sendRedirect(ERROR_PAGE);
35  }
36
```

- In the code, the programmer writes code to send **error page** if user **modifies GET requests**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Cross-site Request Forgery (CSRF) Countermeasures

1. Web applications should use strong **authentication methods** such as cookies, http authentication, etc.
2. Use page tokens such as **time tokens** that change with every http or https page request
3. Appropriately use **GET** and **POST requests**
4. Check the referrer such as HTTP "**referer**" or **referrer** to mitigate this type of attacks
5. Implement OWASP **CSRFGuard library** that uses synchronizer token patterns to minimize the risk of CSRF attacks

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Directory Traversal

- Directory traversal is an HTTP exploit where an attacker can **access restricted files** and **directories** stored on the web server by bypassing the validation process
- Attackers can manipulate variables of **absolute** or **relative path names** that contain file links or can use "**../**" sequences with file names and access files and directories

Insecure code

```
1  import java.io.File;
2  import java.nio.file.Path;
3  import java.nio.file.Paths;
4
5  public class DirectoryTraversal {
6
7      public static void main(String args[])
8      {
9          File f1=new File(System.getProperties()+System.getProperty("file.separator")+args[0]);
10         String absolpath = f1.getAbsolutePath();
11         if (!isInSecureDir(Paths.get(absolpath))) {
12             throw new IllegalArgumentException();
13         }
14         if (!validate(absolpath)) {
15             // Validation Check
16             throw new IllegalArgumentException();}
```

- The code snippet uses command line argument for file path and uses **File.getAbsolutePath()** method to find the absolute path

Secure Code

```
import java.io.File;
import java.nio.file.Path;
import java.nio.file.Paths;

public class DirectoryTraversal {

    public static void main(String args[])
    {
        File f1=new File(System.getProperties()+System.getProperty("file.separator")+args[0]);
        String canPath = f1.getCanonicalPath();

        if (!isInSecureDir(Paths.get(canPath))) {
            throw new IllegalArgumentException();
            }

        if (!validate(canPath)) {
            // Validation Check
            throw new IllegalArgumentException();}
```

- The **getCanonicalPath()** method is used to resolve any aliases, symbolic links and shortcuts

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Directory Traversal Countermeasures

1. Use **strong filter** mechanisms for user data
2. Use filters for form input such as **GET** and **POST** parameters
3. Update **server software** at regular intervals
4. Define access rights to the **protected areas** of the website
5. Apply **checks/hot** fixes to prevent exploitation

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

HTTP Response Splitting

- HTTP response splitting is a web application vulnerability that can compromise an application and its environment with input values mostly through **HTTP requests**
- This attack creates **CR (Carriage Return)** and **LF (Line Feed)** sequences in the **HTTP header** and controls the remaining header and body of the response that the application intends to send and also can create another HTTP message
- Attacker splits the HTTP response by:
  - HTTP Header Splitting
  - HTTP redirect
  - HTTP cookie header

Insecure Code

UserLogin.java

```
response.sendRedirect(Const.TASK_SERVLET_URL +"?actionlink=" + Const.GOTO_FIELD +"&" + Const.FIELD_ID);
```

- The code shows **cookie header response** split

Secure Code

```
1: HTTP/1.1 200 OK
2: (etc…)
3: Set-Cookie: username=ECuser
4: HTTP/1.1 200 OK
5: (malicious content here)
```

- The code uses CR/LF to prevent any HTTP response splitting attack

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

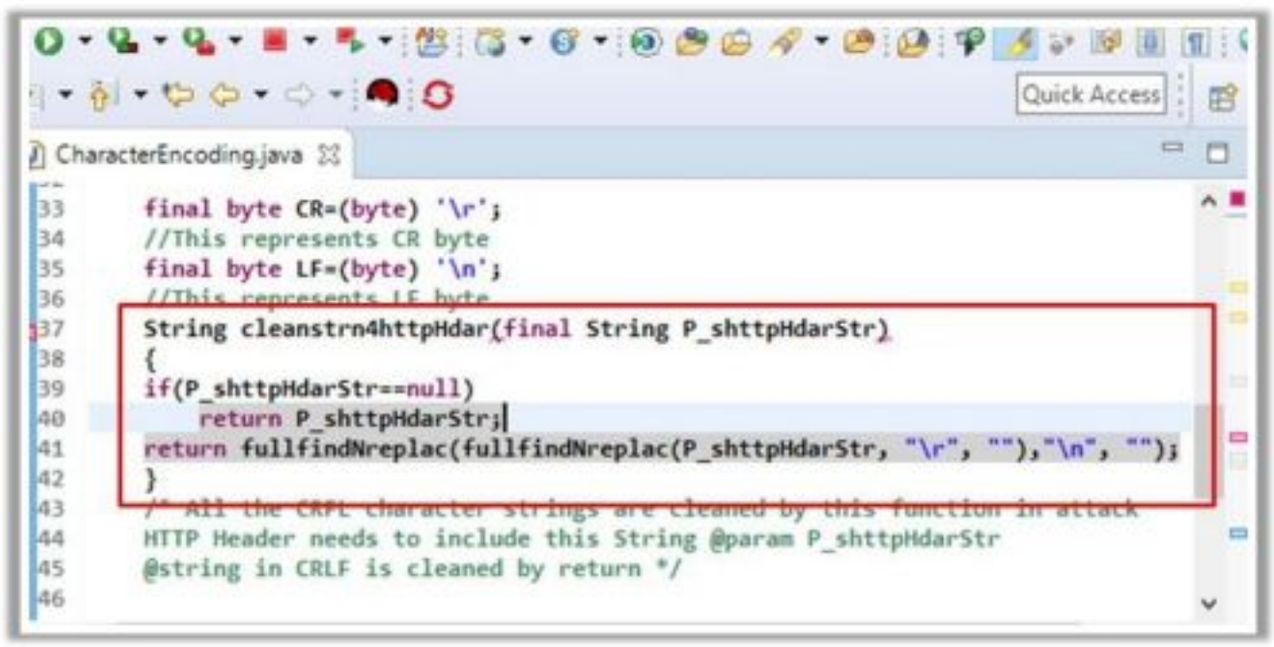
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

HTTP Response Splitting Countermeasures

- Avoid direct placing of user **input variables** in the HTTP header
- Disallow and Filter **CR**/**LF** characters
- Use encoding for **validating header input** and output

Example: Secure Code for HTTP response Splitting Attack

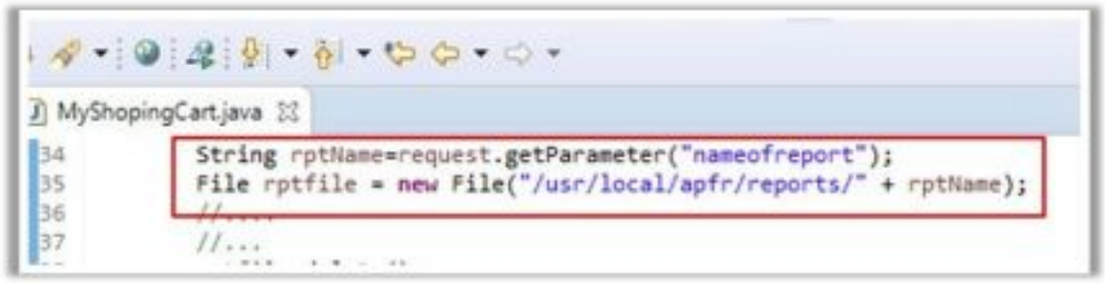


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Parameter Manipulation and Countermeasures

- Use strong **input validating mechanisms** for user data inputs
- Implement strict **application security** routines and updates
- Implement **standards** for minimum and maximum allowable length, characters, patterns and numeric ranges
- Use strictly configured **firewalls** to block and **identify parameters** that are defined in a web page

Insecure Code



- An attacker can **manipulate** the file name or **delete** desired one

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Protecting Application from Log Injection Attack

- Unsanitized user input may leak sensitive data of the application
- **Log Entry** may be split by attackers with a character of carriage return or line feed that might mislead the auditor

Vulnerable Code

```
28            else if (loginCorrect) {
29
30           logger.severe("User login succeeded for: " + username);
31        }
32            else {
33           logger.severe("User login failed for: " + username);
34        }
35        }
36
37    }
```

Secure Code

```
            else if (loginCorrect) {

           logger.severe("User login succeeded for: " +sanitizeUser(usernam
        }
            else {
           logger.severe("User login failed for: " +sanitizeUser(username))
        }
        }

public static String sanitizeUser(String username) {
42
43      return Pattern.matches("[A-Za-z0-9_]+", username) ? username :"unauthorized
```

- If users do not include secure sanitization then it is vulnerable to Log injection
- In the code whitelisting shows only character, number and space allowed

Log Containing Misleading Data

```
java.util.logging.LogManager$RootLogger log
SEVERE: login for: guest
java.util.logging.LogManager log
SEVERE: Login for: administrator
```

- There are different ways of handling log injection attacks
  1. Whitelisting
  2. Blacklisting

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

XML Injection

- XML injection is an **attack technique** where an attacker injects **XML Tags** in the SOAP message to change the structure of XML

Insecure Code

```
package Validation;
import java.io.BufferedOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.util.regex.Pattern;

public class ValidateUser {

    OutputStream outputstream;
    private void createXMLStream(BufferedOutputStream bufOpStrm, String quantity)
            throws IOException {

        String xmlStr = "<item>\n<description>Widget</description>\n" +
        "<price>500</price>\n" +"<quantity>"+quantity+"</quantity></item>";

        outputstream.write(xmlStr.getBytes());
        outputstream.flush();
        }
    public void test() throws IOException
    {
        createXMLStream(null, null);
    }
```

Secure Code

```
1  package Validation;
2  import java.io.BufferedOutputStream;
3  import java.io.IOException;
4  import java.io.OutputStream;
5  import java.util.regex.Pattern;
6
7  public class ValidateUser {
8
9      OutputStream outputstream;
10     private void createXMLStream(BufferedOutputStream bufOpStrm, String quantity)
11             throws IOException {
12
13         if(!Pattern.matches("[0-9]+", quantity)) {
14
15         }
16         String xmlStr = "<item>\n<description>Widget</description>\n" +
17         "<price>500</price>\n" +"<quantity>"+quantity+"</quantity></item>";
18
19         outputstream.write(xmlStr.getBytes());
20         outputstream.flush();
21         }
22     public void test() throws IOException
```

- This code uses string **concatenation** to build an XML query, but **fails** to **validate** the input resulting in an XML attack
- In the code, a **definite number** (0-9) is assigned by the coder to validate inputs

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Command Injection

- Command injection is an attacking method in which a hacker **alters the contents** of the webpage by using HTML code into the input mechanism and by identifying the form fields that lack valid constraints

Insecure Code

```
14
15
16
17 String bktype = request.getParameter("backuptype");
18 String commnd = new String("commnd.exe /k\"c:\\util\\myDb.bat"+ bktype + "&&c:\\utl\\cleanup.bat\"");
19 Runtime.getRuntime().exec(commnd);
```

- In the code, attackers can execute **multiple commands** separated by two ampersands passing a string "**&& del c:\\dbms*.***"

Secure Code

```
124
125
126    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletExc
127        final int Maxsname = 50;
128         final String f_reglrexp="[a-zA-z]{1,"+Maxsname+"}";
129         final Pattern bkpattrn = Pattern.compile(f_reglrexp);
130         final String bkname=f_reglrexp;
131         ValdtBkname(bkname,bkpattrn);
132
133     String bktype = request.getParameter("backuptype");
134     String commnd = new String("commnd.exe /k\"c:\\util\\myOb.bat"+ bktype + "&&c:\\utl\\cleanup.bat\
135     Runtime.getRuntime().exec(commnd);
136         doGet(request, response);
137     }
```

- The code snippet uses **whitelisting to prevent command injections**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

LDAP Injection

- An LDAP (Lightweight Directory Access Protocol) injection attack is done by exploiting the vulnerabilities of **input validation** to run arbitrary LDAP statements against information directories
- In the code snippet, **`searchRecord()`** method is used for searching the directory using LDAP protocol. However, the string filter used in the method does not perform adequate filtering resulting in executing specially crafted inputs

Insecure Code

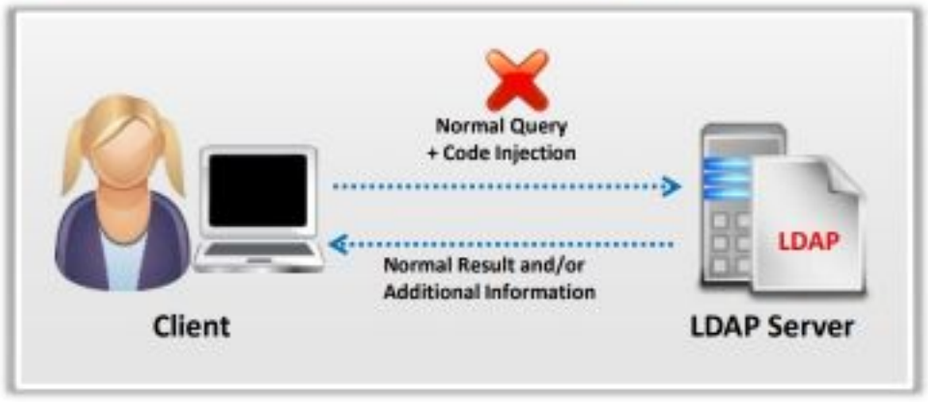
```
5  import javax.naming.NamingException;
6  import javax.naming.directory.Attribute;
7  import javax.naming.directory.Attributes;
8  import javax.naming.directory.DirContext;
9  import javax.naming.directory.InitialDirContext;
10 import javax.naming.directory.SearchControls;
11 import javax.naming.directory.SearchResult;
12 import javax.xml.xpath.XPathExpression;
13
14 public class path {// String UsrsName = "S*"; // Invalid
15     // String Usrpwd = "*"; // Invalid
16
17     private void searchRecord(String UsrsName, String Usrpwd) throws NamingExcep
18     Hashtable<String, String>envrnmnt = new Hashtable<String, String>();
19     envrnmnt.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFact
20     try {
21     DirContext dirctx = new InitialDirContext(envrnmnt);
22     SearchControls srchctrl = new SearchControls();
23     String[] attrbfltr = {"cnt", "email"};
24     srchctrl.setReturningAttributes(attrbfltr);
25     srchctrl.setSearchScope(SearchControls.SUBTREE_SCOPE);
26     String bsc = "dircon=example, dircon=com";
27     // This results to (&(strn=S*)(Usrpwd=*))
28     String fltr = "(&(strn=" + UsrsName + ")(Usrpwd=" + Usrpwd + "))";
29     NamingEnumeration<?> rslts = dirctx.search(bsc, fltr, srchctrl);
30     while (rslts.hasMore()) {
31     SearchResult schrs = (SearchResult) rslts.next();
32     Attributes atrbs = schrs.getAttributes();
33     Attribute attr = atrbs.get("cnt");
34     System.out.println(attr.get());
35     attr = atrbs.get("email");
36     System.out.println(attr.get()); }
37     dirctx.close();
38     }
39     catch(Exception e) {
40
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

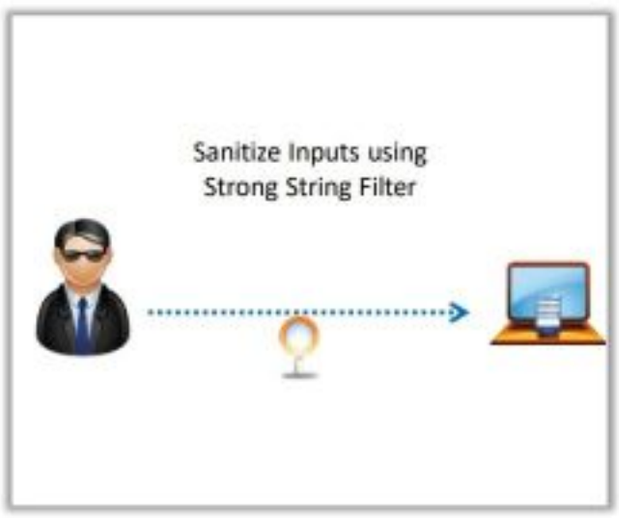
@ErrorCatchers

LDAP Injection (Cont'd)





Secure Code



path.java

```
9
10    public static XPathExpression compile(String string) {
11
12
13        String UsrsName = "Sherlock Holmes"; // Valid
14        String Usrpwd = "secret2";// Valid
15
16        srchctrl.setSearchScope(SearchControls.SUBTREE_SCOPE);
17        String bsc = "dircon=example, dircon=com";
18        if(!UsrsName.matches("[\\w\\s]*") || !Usrpwd.matches("[\\w]*")) {
19        throw new IllegalArgumentException("Invalid input");
20        }
21        String fltr = "(&(strn = " + UsrsName + ")(userPassword=" + Usrpwd + "))
22
```

- The code uses **whitelisting technique** to sanitize inputs using strong String filter that contains valid characters

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

XML External Entity Attack

1. An **XML External Entity attack** enables the attacker to access local or remote contents resulting in information disclosure and denial of service attacks
2. The external entities use SYSTEM Keyword; the XML parser reads the document from the URI specified
3. This value can be replaced by the attacker to read any local or remote file



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

XML External Entity Attack (Cont'd)

- Prevent **XML Extrenal Entity attacks** by implementing org.xml.sax.EntityResolver resolveEntity() method
- Contains customized handler using whitelist for checking external entites
- resolveEntity() method resolves the external entities of XML document while parsing

Insecure Code

FileValidation.java | XmlValidation.java

```
1  import java.io.FileInputStream;
2  import java.io.IOException;
3  import java.io.InputStream;
4
5  import javax.xml.parsers.ParserConfigurationException;
6  import javax.xml.parsers.SAXParser;
7  import javax.xml.parsers.SAXParserFactory;
8
9  import org.xml.sax.helpers.DefaultHandler;
10
11 import jdk.internal.org.xml.sax.SAXException;
12 import jdk.nashorn.internal.runtime.ParserException;
13
14 public class XmlValidation {
15
16     private static DefaultHandler DefaultHandler;
17
18     public static void recivexmlStream(InputStream inStream, DefaultHandler defaultHandler)
19             throws ParserException, SAXException, IOException, org.xml.sax.SAXException {
20
21         SAXParserFactory factory = SAXParserFactory.newInstance();
22         SAXParser saxParser = null;
23         try {
24             saxParser = factory.newSAXParser();
25         } catch (ParserConfigurationException e) {
26             // TODO Auto-generated catch block
27             e.printStackTrace();
28         } catch (org.xml.sax.SAXException e) {
29             // TODO Auto-generated catch block
30             e.printStackTrace();
31         }
32         saxParser.parse(inStream, defaultHandler);
33     }
34
35     public static void main(String args[]) throws ParserException, SAXException, IOException {
36         try {
37             recivexmlStream(new FileInputStream("evil.xml"), DefaultHandler);
38
39         } catch (Exception e) {
40             // TODO: handle exception
41         }
42
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

XML External Entity Attack (Cont'd)

Secure Code

- The below code snippet implements **org.xml.sax.EntityResolver** resolveEntity() **method**

FileValidation.java | XmlValidation.java | SecureCustomResolver.java

```
1  import java.io.IOException;
2
3  import org.xml.sax.EntityResolver;
4  import org.xml.sax.InputSource;
5  import org.xml.sax.SAXException;
6
7  public class SecureCustomResolver implements EntityResolver
8  {
9
10     @Override
11     public InputSource resolveEntity(String publicId, String systemId) throws SAXException, IOException
12     {
13
14         String entityPath="file:/Users/onlinestore/good.xml";
15         if(systemId.equals(entityPath)) {
16             System.out.println("Resolving entity: "+publicId +""+ systemId);
17             return new InputSource(entityPath);
18             }else
19             {
20                 return new InputSource();
21             }
22 }
23 }
```

resolveEntity() method Syntax

```
InputSource resolveEntity(String publicId, String systemId) throws SAXException, IOException
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Unrestricted File Upload Attack

- The **File Upload** feature exposes the application to upload malicious files. This results in both, **client side** and **server side** attacks
- **While uploading files restrict**
  - Uploading harmful content like exe
  - Overwriting existing files
  - Uploaded file size to prevent Denial of Service attack

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Prevent Unrestricted File Upload: Validate File Extension

Secure Code

- The code **validates file extension** using **regular expression**

FileValidation.java

```
1  import java.util.regex.Matcher;
2  import java.util.regex.Pattern;
3
4  public class FileValidation {
5
6
7      private static Pattern fileExtnPtrn = Pattern.compile("([^\\s]+(\\.(?i)(txt|doc|csv|pdf))$)");
8
9      public static boolean validateFileExtn(String userName){
10
11         Matcher mtch = fileExtnPtrn.matcher(userName);
12         if(mtch.matches()){
13             return true;
14         }
15         return false;
16     }
17     public static void main(String a[]){
18         System.out.println("Is 'java2novice.pdf' allowed file? "
19                         +validateFileExtn("java2novice.pdf"));
20         System.out.println("Is 'cric.doc' allowed file? "
21                         +validateFileExtn("cric.doc"));
22         System.out.println("Is 'java.gif' allowed file? "
23                         +validateFileExtn("java.gif"));
24         System.out.println("Is 'novice.mp3' allowed file? "
25                         +validateFileExtn("novice.mp3"));
26         System.out.println("Is 'java_2.jpeg' allowed file? "
27                         +validateFileExtn("java_2.jpeg"));
28     }
29
30 }
31
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Injection Attacks Countermeasures

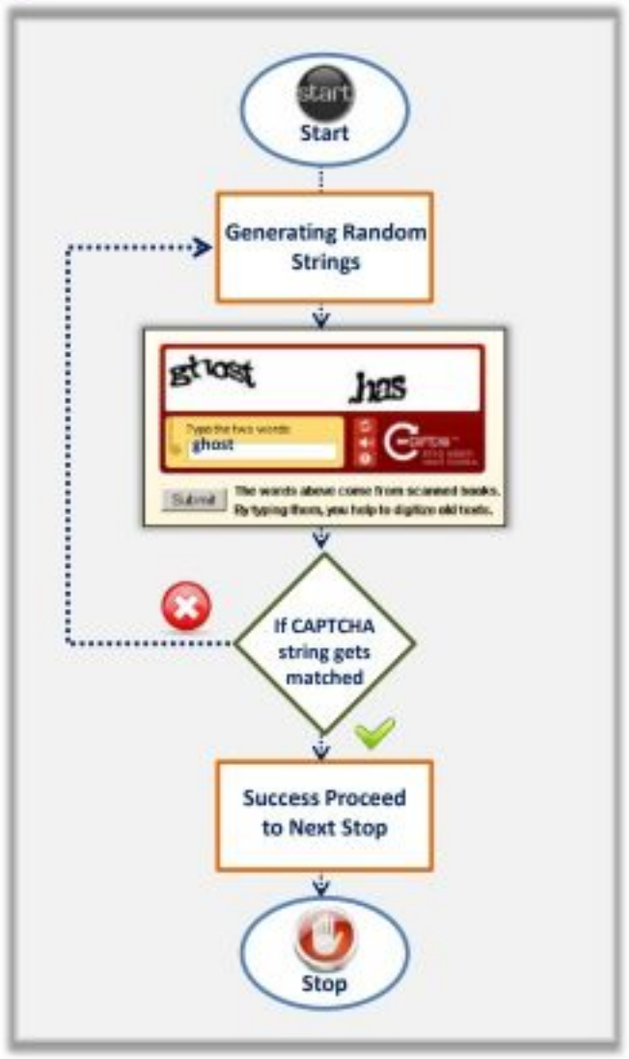
Avoid implementing **custom parsers** as there are chances of ignoring **attack vectors.** So it is always better to use **tested** and **secured** parsers
While using **DOM** based parser, validate the XML stream size against a maximum before parsing
Defend against Denial of Service attacks by using **SAX** based parsing
Validate input to find the presence of malicious characters to avoid **XML injection attacks**
A thorough **input validation** needs to be performed before a request is sent to the **database**
Differentiate input strings from executable commands by using **parameterized stored procedures** to access the database or use **SQL parameters** to build SQL commands
It is always **suggested** to use less **privileged** accounts to access the database
Replace all **single quotes** with two single quotes
Error reports should not be displayed to the outsiders or else the same errors are used by attackers for **further attacks** / **malicious purpose**
Database server and **webserver** should be isolated and located on different machines

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

CAPTCHA

- CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart)

- Used to **authenticate** user input
- Consists of randomly generated image with **distorted letters/numbers**
- Determines whether the input is given by a **real user** or a computer program
- Used to **mitigate attacks** caused by automated tools
- Implemented on applications using third party software like **jcaptcha**, **simplecaptcha**, **recaptcha**



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Sample Code for Creating CAPTCHA

The creation of CAPTCHA is given in the Servlet codeCaptchaServlet.java



Source: *http://techbycaltech.com*

```
Login.jsp  index.jsp  result.jsp  CaptchaVerif...  CaptchaServl...
4  import java.io.OutputStream;
5  import java.util.Random;
6
7  import javax.imageio.ImageIO;
8  import javax.servlet.ServletException;
9  import javax.servlet.annotation.WebServlet;
10 import javax.servlet.http.HttpServlet;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13 import java.awt.Color;
14 import java.awt.Font;
15 import java.awt.GradientPaint;
16 import java.awt.Graphics2D;
17 import java.awt.RenderingHints;
18 import java.awt.image.BufferedImage;
19
20
21 /**
22  * Servlet implementation class CaptchaServlet
23  */
24 @WebServlet("/CaptchaServlet")
25 public class CaptchaServlet extends HttpServlet {
26     private static final long serialVersionUID = 1L;
27
28     protected void executeReq(HttpServletRequest request,
29             HttpServletResponse response) throws ServletException, IOException {
30             final int wd = 200;
31             final int ht = 50;
32             char cdata[][] = new char[1][];
33             cdata[0] = getRandomNumber(8).toCharArray();
34             final BufferedImage buffrdImage = new BufferedImage(wd, ht,
35             BufferedImage.TYPE_INT_RGB);
36             final Graphics2D G2D = buffrdImage.createGraphics();
37             final Font f = new Font("calibri", Font.BOLD, 18);
38             RenderingHints rHints = new RenderingHints(RenderingHints.KEY_ANTIALIASING,
39             RenderingHints.VALUE_ANTIALIAS_ON);
40             rHints.put(RenderingHints.KEY_RENDERING,
41             RenderingHints.VALUE_RENDER_QUALITY);
42             G2D.setRenderingHints(rHints);
43             G2D.setFont(f);
44             G2D.setColor(new Color(255, 255, 0));
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Sample Code for Creating CAPTCHA (Cont'd)

```
Login.jsp  index.jsp  result.jsp  CaptchaVerif...  CaptchaServl...
49             final GradientPaint grad_Paint = new GradientPaint(0, 0, Color.black, 0, ht / 2,
50                     Color.black, true);
51             G2D.setPaint(grad_Paint);
52             G2D.fillRect(0, 0, wd, ht);
53             G2D.setColor(new Color(255, 153, 0));
54             Random rndno = new Random();
55             final String captcha = String.copyValueOf(cdata[0]);
56             request.getSession().setAttribute("captcha", captcha);
57             int xCordinate = 0;
58             int yCordinate = 0;
59             for (int a = 0; a < cdata[0].length; a++) {
60             xCordinate += 10 + (Math.abs(rndno.nextInt()) % 15);
61             if (xCordinate >= wd - 5) {
62             xCordinate = 0;
63             }
64             yCordinate = 20 + Math.abs(rndno.nextInt()) % 20;
65             G2D.drawChars(cdata[0], a, 1, xCordinate, yCordinate);
66             }
67             G2D.dispose();
68             response.setContentType("image/png");
69
70
71             final OutputStream outStr = response.getOutputStream();
72             ImageIO.write(buffrdImage, "png", outStr);
73             outStr.close(); }
74             protected void doGet(HttpServletRequest request,
75             HttpServletResponse response) throws ServletException, IOException {
76             executeReq(request, response);
77             }
78             protected void doPost(HttpServletRequest request,
79             HttpServletResponse response) throws ServletException, IOException {
80             executeReq(request, response); }
81             public static String getRandomNumber(int length) {
82             String chcode = "abcdefghjijkmnpqrstuvwxyzABCDEFGHJKLMNPQRSTUVWXYZ12345678"
83             Random rand1 = new Random();
84             char[] buf = new char[length];
85             for (int i = 0; i < length; i++) {
86             buf[i] = chcode.charAt(rand1.nextInt(chcode.length()));
87             }
88             return new String(buf);
89
```

Source: *http://techbycaltech.com*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Sample Code for CAPTCHA Verification

UserValidat... | applicationC... | UserLogin.java | Login.jsp | index.jsp | result.jsp | CaptchaVarif...

```
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.annotation.WebServlet;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9 @WebServlet("/Captcha")
10 public class CaptchaVarification extends HttpServlet {
11     private static final long serialVersionUID = 1L;
12
13
14     public CaptchaVarification() {
15         super();
16
17     }
18
19     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
20
21
22         final String vCaptcha = (String) request.getSession().getAttribute("captcha");
23         final String code = (String) request.getParameter("code");
24         if (vCaptcha != null && code != null) {
25         if (vCaptcha.equals(code)) {
26         request.setAttribute("result","Congratulations, You Passed The Captcha Test");
27         } else {
28         request.setAttribute("result",
29         "Sorry, You Failed The Captcha Test");
30         }
31         request.getRequestDispatcher("/result.jsp").forward(request,
32         response);
33
34
35     }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Sample Code for Displaying CAPTCHA

The CAPTCHA is displayed using JSP

index.jsp

UserValidat... | UserLogin.java | Login.jsp | index.jsp

```
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5 <title>Sample Captcha</title>
6 </head>
7 <body>
8
9 <form method="post" action="verifyCaptcha.do">
10 <table >
11 <tr>
12 <td align="center">
13 <img src="samplecaptchaServlet.do">
14 <br />
15 <br />
16 <input type="text" name="capatchacode">
17 </td>
18 </tr>
19 </table>
20 <input type="submit" value="submit">
21 </form>
22 <br>
23 <br>
24
25 </body>
26 </html>
27
```



result.jsp

UserValidat... | Login.jsp | index.jsp | result.jsp

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www
4 <html>
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7 <title>Sample Captcha</title>
8 </head>
9 <body>
10 <%=request.getAttribute("result")%>
11 </body>
12 </html>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Best Practices for Input Validation

Golden Rule: All external inputs, no matter what they are, should be examined and validated
Do not log unvalidated user inputs, as it may result in leaking sensitive data
Ensure that a data validation mechanism is present
Ensure that the data validation occurs on the server side
Ensure that proper length checks on all inputs exist
Examine where data validation occurs, and if a centralized model or decentralized model is used
Ensure that all fields, cookies, HTTP headers/bodies & form fields are validated
Ensure that there are no backdoors in the data validation model

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Best Practices for Input Validation (Cont'd)

Apply known good values, i.e., whitelists, wherever it is feasible
Do not incorporate input strings of untrusted sources into format strings
Ensure that all inputs are canonicalized
Perform filtering and validation mechanisms based on the strings' character data
Ensure that inputs follow correct syntax and content restrictions
Use java.sql.PreparedStatement class to prevent SQL injection attacks
Ensure that all inputs that can (and will) be modified by a user such as HTTP headers, input fields, hidden fields, etc., are properly validated
Ensure that the data is well-structured and contains only known good characters, if possible

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Module Summary

- Input data validation is most important for any software application because of the dangers posed by tainted input at various levels of its execution
- Whitelisting is a process to verify the inputs against the possible expected known values
- Testing input values against a set of negative expected inputs is called as blacklisting
- Java framework allows controlling user behavior in Java and helps save time by eliminating the creation of test scripts
- A regular expression provides a concise and flexible means to "match" (specify and recognize) strings of text, such as particular characters, words, or patterns of characters
- The Struts validator framework is used to validate input on both the user browser and the server side
- Prepared statements define methods and properties that are used to send SQL commands to retrieve data from the database
- Stored procedures prevent SQL injection attacks by performing thorough validation on any user input before processing data
- Golden Rule: All external input, no matter what it is, is examined and validated

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

JAVA

C | ASE

Certified | Application Security Engineer



Module 05

Secure Coding Practices for Authentication and Authorization

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

Module Objectives

1	Understand Authentication Concepts
2	Discuss Authentication Implementation in Java
3	Discuss Authentication Weaknesses and Prevention
4	Understand Authorization Concepts
5	Discuss Access Control Model
6	Discuss EJB Authorization
7	Discuss Java Authentication and Authorization (JAAS)
8	Discuss Authorization Common Mistakes and Countermeasures
9	Discuss Java EE Security
10	Discuss Authentication and Authorization in Spring Security Framework
11	Learn Defensive Coding Practices against Broken Authentication and Authorization

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Introduction to Authentication

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Introduction to Authentication

- The main purpose of authentication is to verify a **user's access** to a protected segment of the web application
- Authentication is of two types, namely **basic authentication** and **form-based authentication**

The responsibilities of authentication systems are:

- Verify the user and provide a safe communication passage
- Gather information about the way the user accesses the system

J2EE supports four types of authentication mechanisms

- HTTP Basic Authentication
- Form-based Authentication
- Client/Server Mutual Authentication
- HTTP Digest Authentication

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java Container Authentication

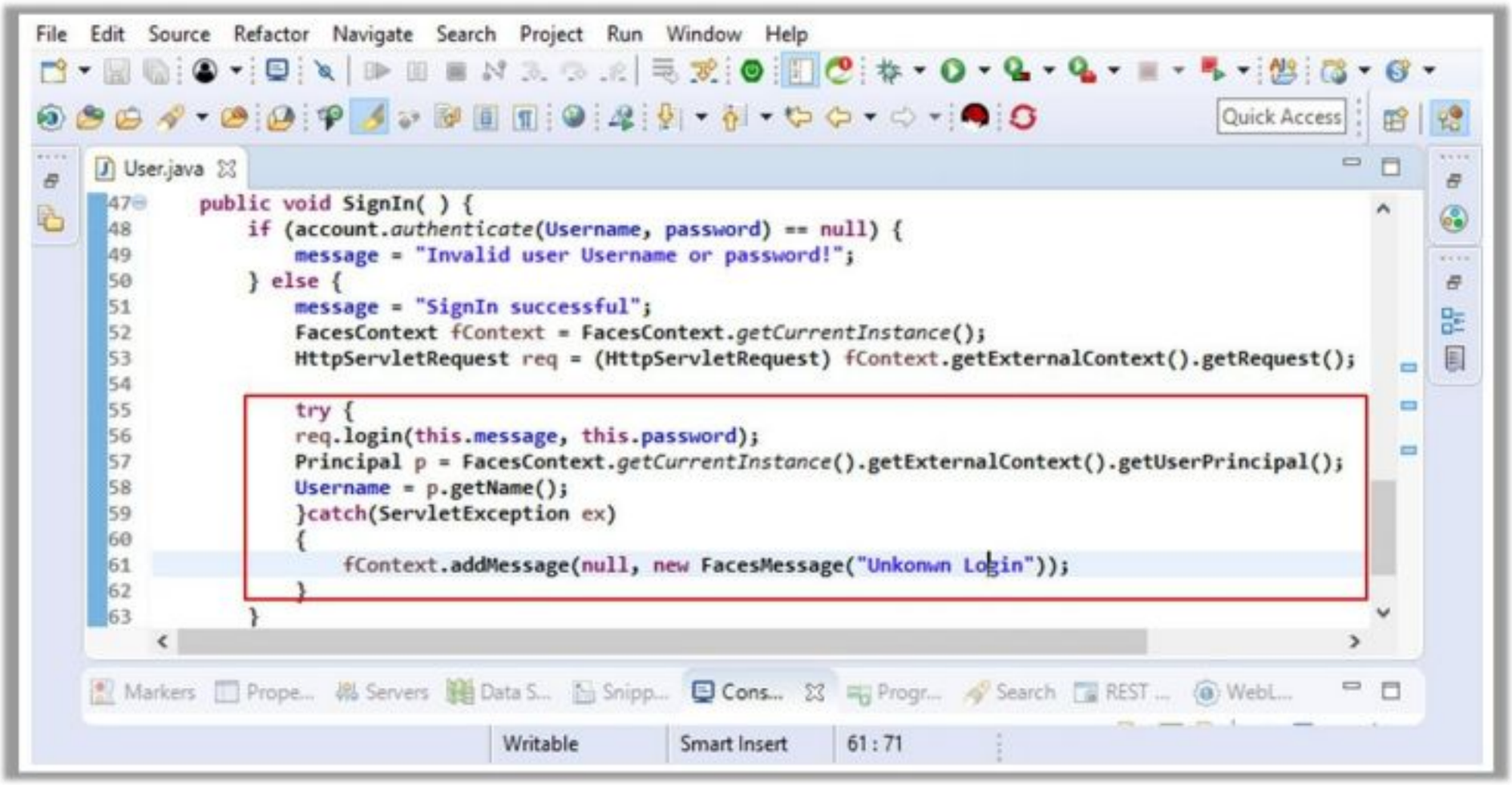
- Authentication in a web application is **role-based**, i.e., a user needs to be assigned a role in order to access the web application, e.g., a customer, developer or manager
- A role gets access only when the identity of the user is identified by the container, and for all other cases, access is denied, resulting in **HTTP 401 status code**
- Information about the role and user that is already stored in the **database** of the web application is compared with that of the user who wants to access the web application
- After **authentic validation** of the user, his roles are assigned by the web application through one of the following methods
  - JDBC Login Module
  - LDAP Login Module
  - Windows Login Module
  - Custom JAAS Login Module
- In Java, container authenticates web application, i.e., **tomcat**, **jboss**, performs the process of authentication
- These results of the authentication are passed on to the web application, thus avoiding the need to implement authentication code such as **HTTP Basic** or **NTLM**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Authentication Mechanism Implementation



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Types of Authentication

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

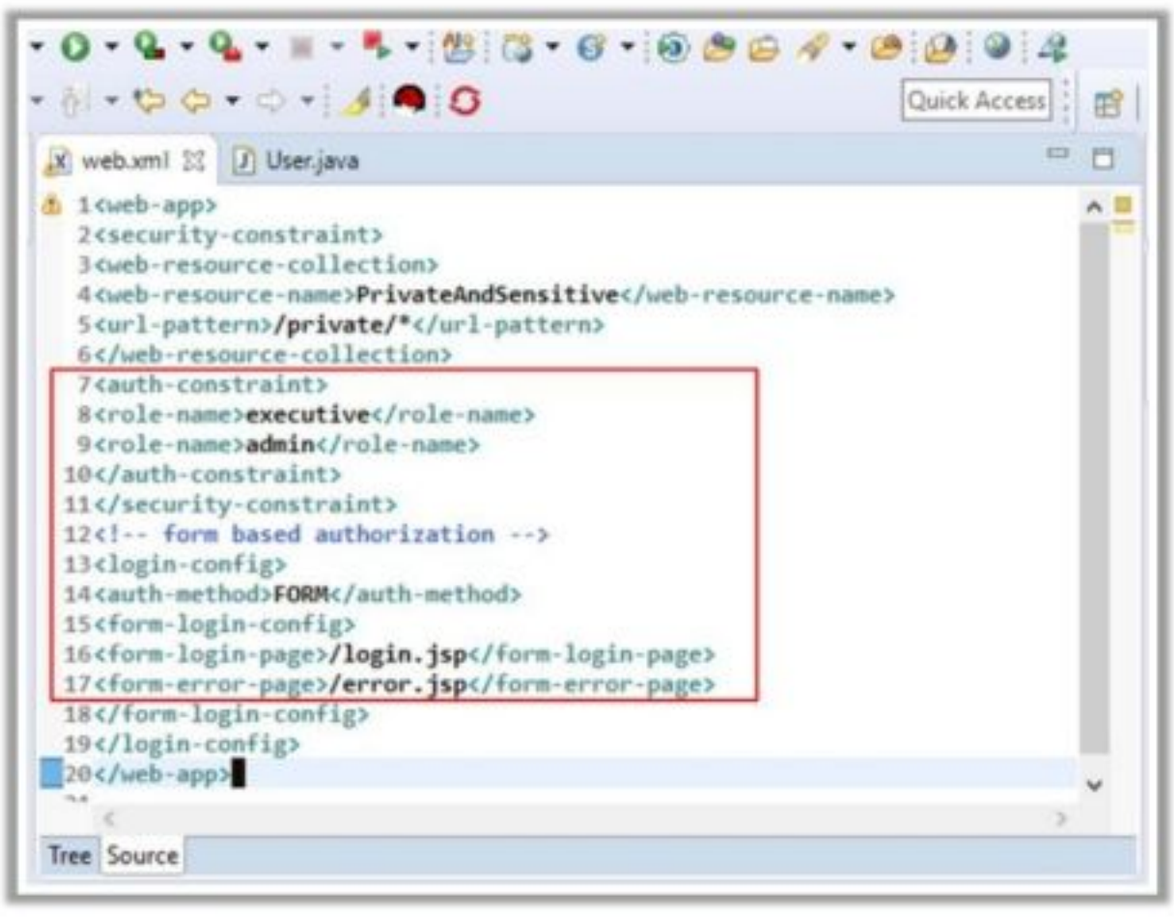
Declarative vs Programmatic Authentication

- Sometimes, declarative security alone is not sufficient; in such cases, programmatic security enhances **security** by giving complex, dynamic **rules** and **policies** for application components
- In programmatic security, business methods that are invoked help J2EE container to determine whether a caller should be given **privileges** or not, i.e., **access** or **denial** of **permissions** to a resource
- Enhanced security can be provided by programmatic mechanism supplementing declarative security for **role membership** and **user identity**
- J2EE container services provide **application tiers** and **components** with authentication and authorization facilities identifying service providers and callers
- J2EE container services support **declarative** and **programmatic authentication**
- Rules and permissions play a key role in declarative security model defined in the deployment descriptor document that is bundled with **application component**
- These **rules** and **permission** are assigned by application deployer according to **deployment descriptor**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Declarative Security Implementation

- **Servlet containers** are used to implement declarative security
- Program control is through deployment descriptor **web.xml file**
- Declarative security along with servlets and JSP give a **robust protection**



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Programmatic Security Implementation

LoginServlet.java

```
1
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5  import javax.annotation.security.DeclareRoles;
6  import javax.servlet.ServletException;
7  import javax.servlet.annotation.WebServlet;
8  import javax.servlet.http.HttpServlet;
9  import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletResponse;
11
12 @DeclareRoles("ECCUser")
13
14 @WebServlet("/LoginServlet")
15 public class LoginServlet extends HttpServlet {
16     private static final long serialVersionUID = 1L;
17     public LoginServlet() {
18         super();
19     }
20
21     protected void processRequest(HttpServletRequest req, HttpServletResponse resp)throws ServletException,IOException
22     {
23
24         resp.setContentType("text/html;charset=UTF-8");
25         PrintWriter pw = resp.getWriter();
26         try {
27         String UsrName = req.getParameter("txtUserName");
28         String passwd = req.getParameter("txtPassword");
29         pw.println("Before Login" + "<br><br>");
30         pw.println("IsUserInRole?.."
31         + req.isUserInRole("ECCUser")+"<br>");
32         pw.println("getRemoteUser?.." + req.getRemoteUser()+"<br>");
33         pw.println("getUserPrincipal?.."
34         + req.getUserPrincipal()+"<br>");
35         }catch (Exception e) {
36
37         }   }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Programmatic Security Implementation (Cont'd)

LoginServlet.java

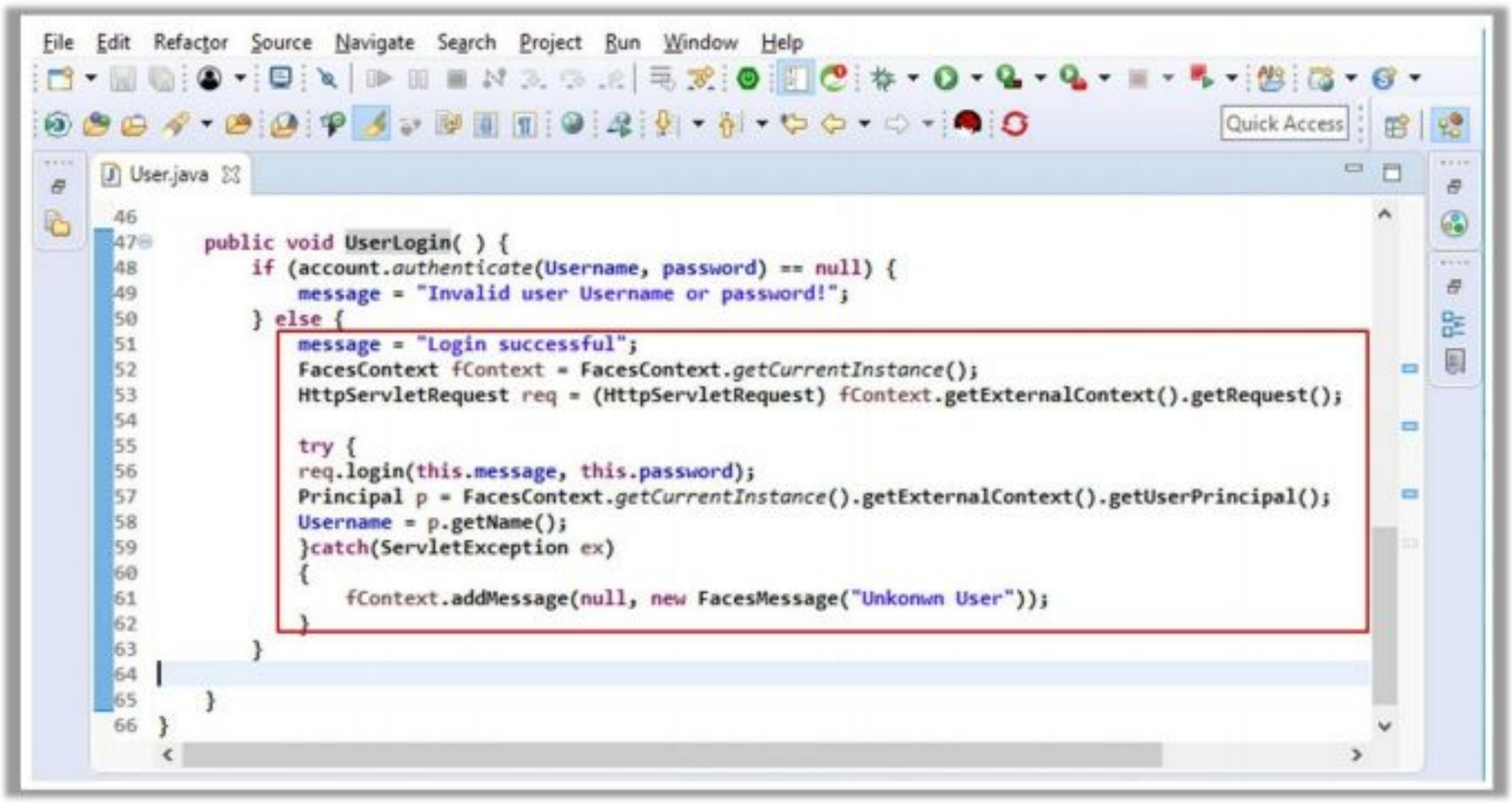
```
35         try {
36         req.login(UsrName, passwd);
37         } catch(ServletException se) {
38         pw.println("Login Failed with a ServletException.."
39         + se.getMessage());
40         return;
41         }
42         pw.println("After Login..."+"<br><br>");
43         pw.println("IsUserInRole?.."
44         + req.isUserInRole("ECCUser")+"<br>");
45         pw.println("getRemoteUser?.." + req.getRemoteUser()+"<br>");
46         pw.println("getUserPrincipal?.."
47         + req.getUserPrincipal()+"<br>");
48         pw.println("getAuthType?.." + req.getAuthType()+"<br><br>");
49         req.logout();
50         pw.println("After Logout..."+"<br><br>");
51         pw.println("IsUserInRole?.."
52         + req.isUserInRole("ECCUser")+"<br>");
53         pw.println("getRemoteUser?.." + req.getRemoteUser()+"<br>");
54         pw.println("getUserPrincipal?.."
55         + req.getUserPrincipal()+"<br>");
56         pw.println("getAuthType?.." + req.getAuthType()+"<br>");
57
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Java EE Authentication Implementation Example



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Basic Authentication

- Basic authentication uses **standard HTTP basic security**
- Authentication is through SSL using dialog box instead of HTML form, requiring no changes in **JSP pages** or **servlets.** SSL/TLS should be explicitly enabled else the credentials will be transmitted in plain text
- Easier to setup since it requires no creation and **configuration** of **login** or **login error pages**
- Mostly, **intranet applications** use basic authentication



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

How to Implement Basic Authentication?

tomcat-users.xml

```
48  <role rolename="engineer" />
49  <role rolename="developer" />
50  <role rolename="tester" />
51  <role rolename="admin" />
52  <user username="john" password="test@123" roles="developer,admin" />
53  <user username="thomas" password="abcdefg" roles="tester,admin" />
54  <user username="joseph" password="dream123" roles="tester,engineer" />
55  <user username="julie" password="123456" roles="developer,admin" />
56 </tomcat-users>
```

- Define **roles** and **setup** users with usernames, passwords, and roles

web.xml

```
21  <security-constraint>
22  </security-constraint>
23 <login-config>
24    <auth-method>BASIC</auth-method>
25    <realm-name>MyRealm</realm-name>
26 </login-config>
27 </web-app>
```

- Activating **BASIC** authentication in the server and designation of **Realm**

tomcat-users.xml | web.xml

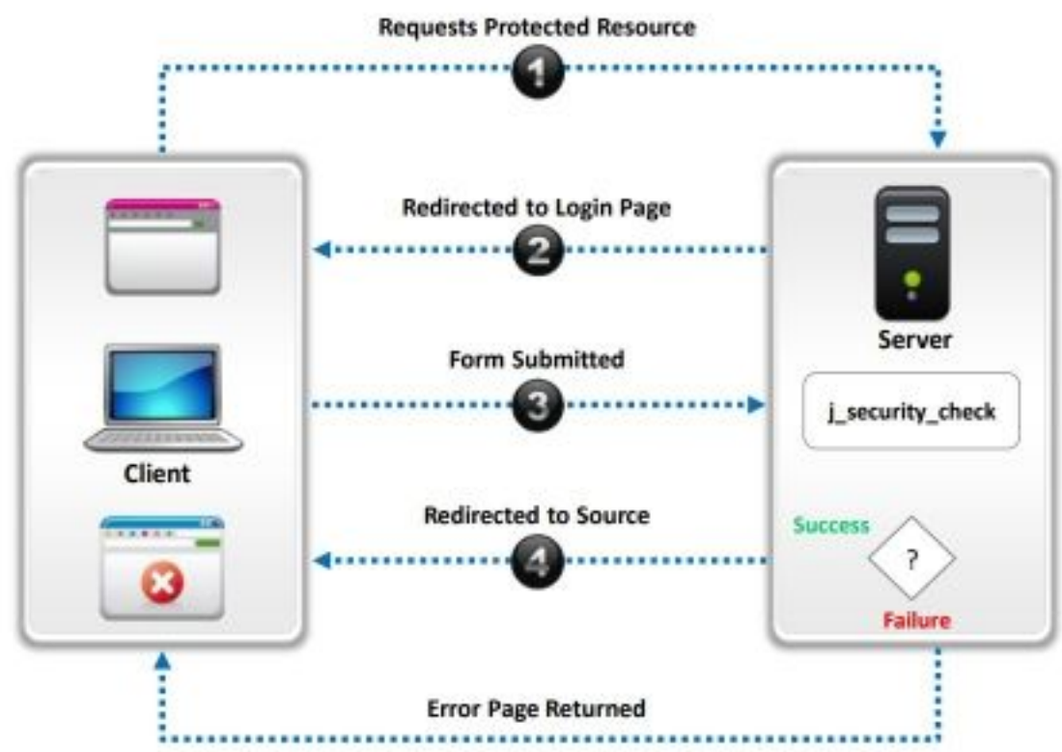
```
39  <display-name></display-name>
40  <user-data-constraint><description></description>
41  <transport-guarantee>CONFIDENTIAL</transport-guarantee
42  </user-data-constraint>
```

- Implementing **BASIC** Authentication with SSL

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Form-based Authentication

- Form-based authentication is an **HTML-based form** that uses declarative security
- Easy implementation through control over **GUI** of the authentication form
- User **credentials** and **roles** are checked for their correctness, if found incorrect, error page appears
- Container handles the entire process of **authentication**
- **SSL** can be added to part or whole of the **web application**

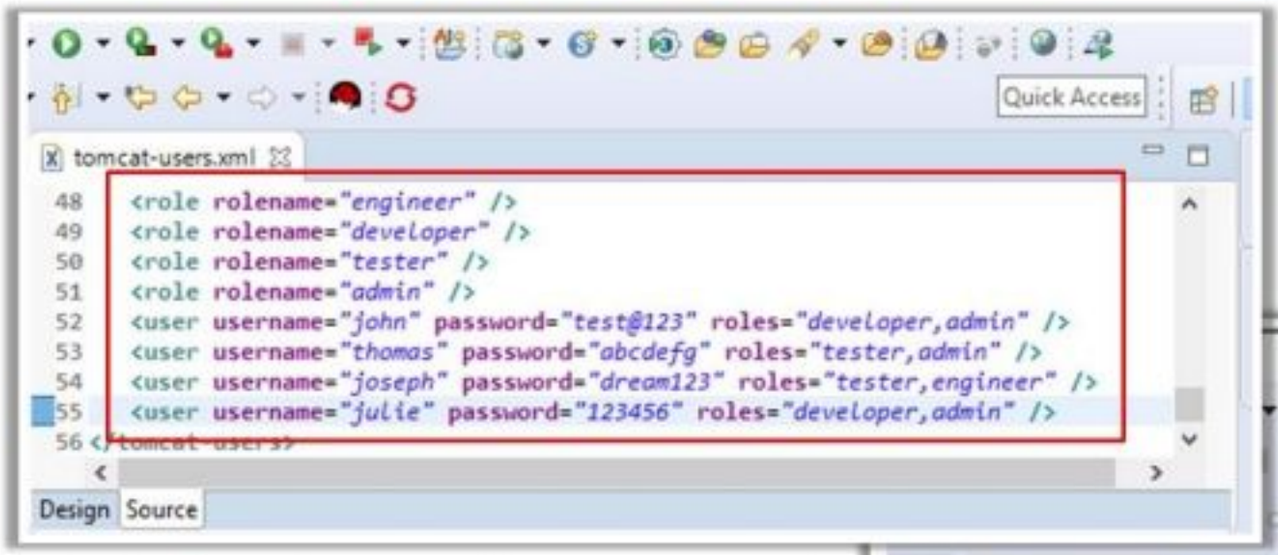


Form-Based Authentication

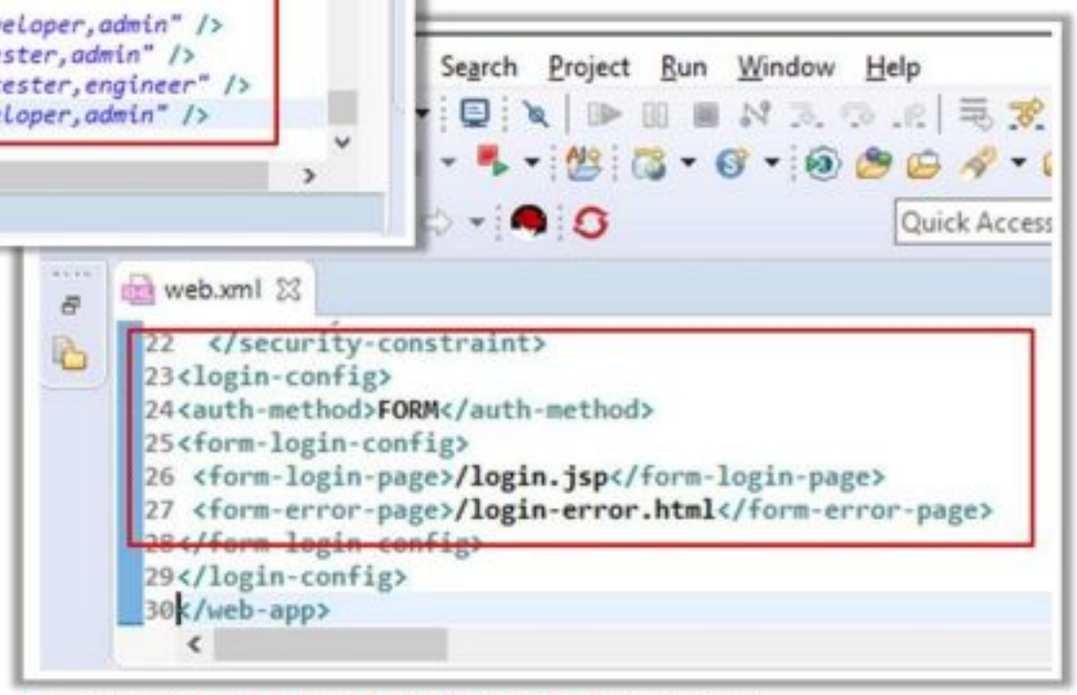
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Form-based Authentication Implementation



- Define **roles** and **setup** users with **usernames**, passwords, and roles



- Activating **Form-based authentication** in the server

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Implementing Kerberos-Based Authentication

- Kerberos V5 is mainly used as an **authentication security protocol** within a domain
- It verifies a user as well as the server providing information; this is known as **mutual authentication**

Working Overview of Kerberos

1. This mechanism **issues tickets** containing user identity, encrypted password, and encrypted data
2. Working process is **invisible** to the **user**
3. Key Distribution Centre in Kerberos **stores** account **information** and client **passwords**

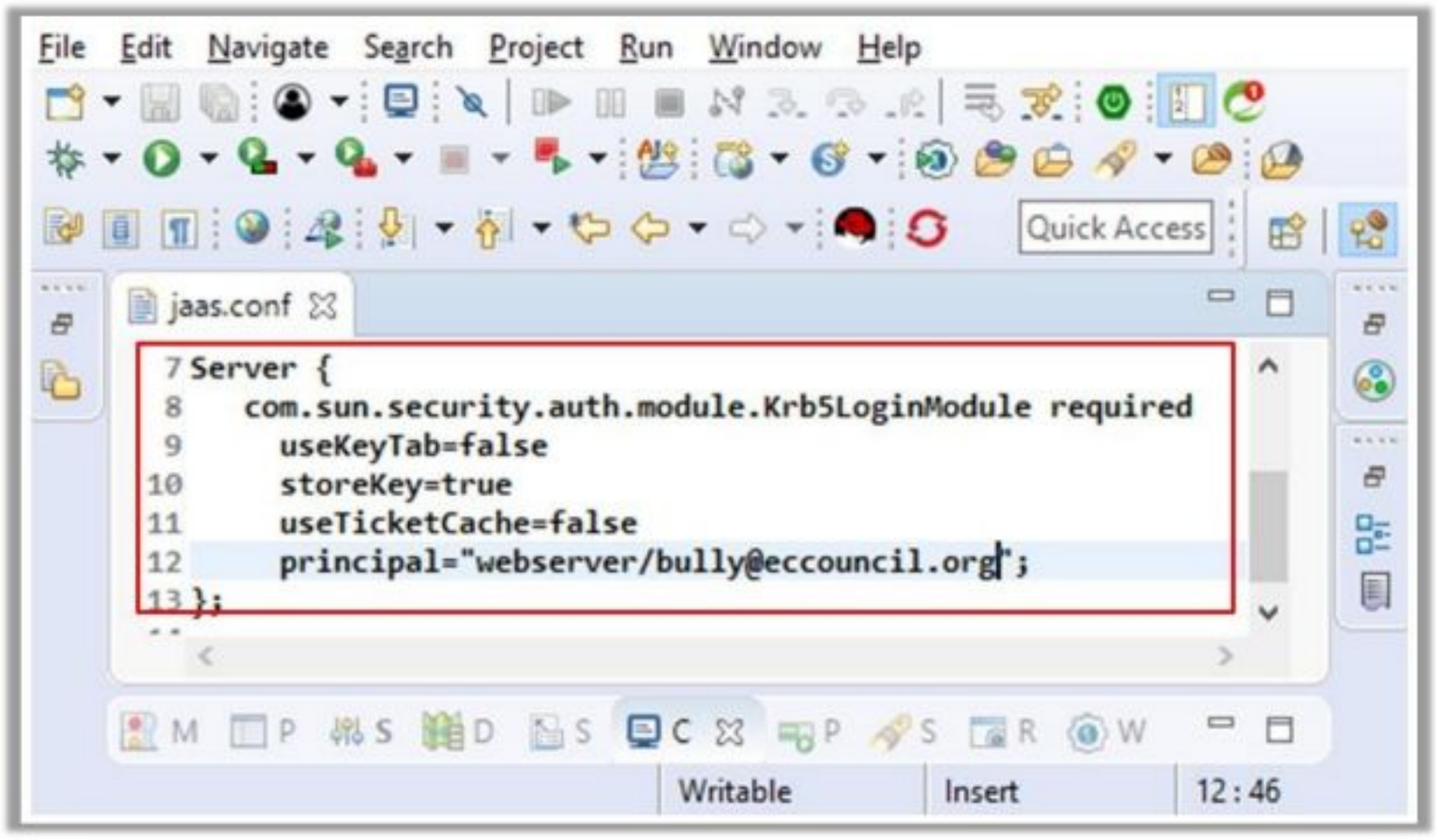
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secured Kerberos Implementation

Example for Kerberos Implementation with jaas.conf



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secured Kerberos Implementation (Cont'd)

- Implementing Kerberos with ***CallbackHandler.java**
- LoginCallbackHandler implements **CallbackHandler**

LoginCallbackHandler.java

```
1  package Ecouncil.app.gss;
2  import javax.security.auth.callback.Callback;
7
8  /**
9   * Password callback handler for resolving password/usernames for a JAAS login.
10  * @author Ants
11  */
12 public class LoginCallbackHandler implements CallbackHandler {
13
14   public LoginCallbackHandler() {
15     super();
16   }
17 private String CompanyUId;
18 private char [] CompanyPasswd;
19 public void handle(Callback[] callbks)
20 throws java.io.IOException, UnsupportedCallbackException {
21 for (int i = 0; i < callbks.length; i++) {
22 if (callbks[i] instanceof NameCallback) {
23 NameCallback Ncb = (NameCallback)callbks[i];
24 Ncb.setName(CompanyUId);
25 } else if (callbks[i] instanceof PasswordCallback) {
26 PasswordCallback Ncb = (PasswordCallback)callbks[i];
27 Ncb.setPassword(CompanyPasswd);
28 } else {
29 throw new UnsupportedCallbackException(callbks[i]);
30 } } }
31 public void setUserId(String UsrId)
32 {
33 CompanyUId = UsrId;
34 }
35 public void setPassword(String pwd)
36 {
37 CompanyPasswd = pwd.toCharArray();
38 } }
39
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

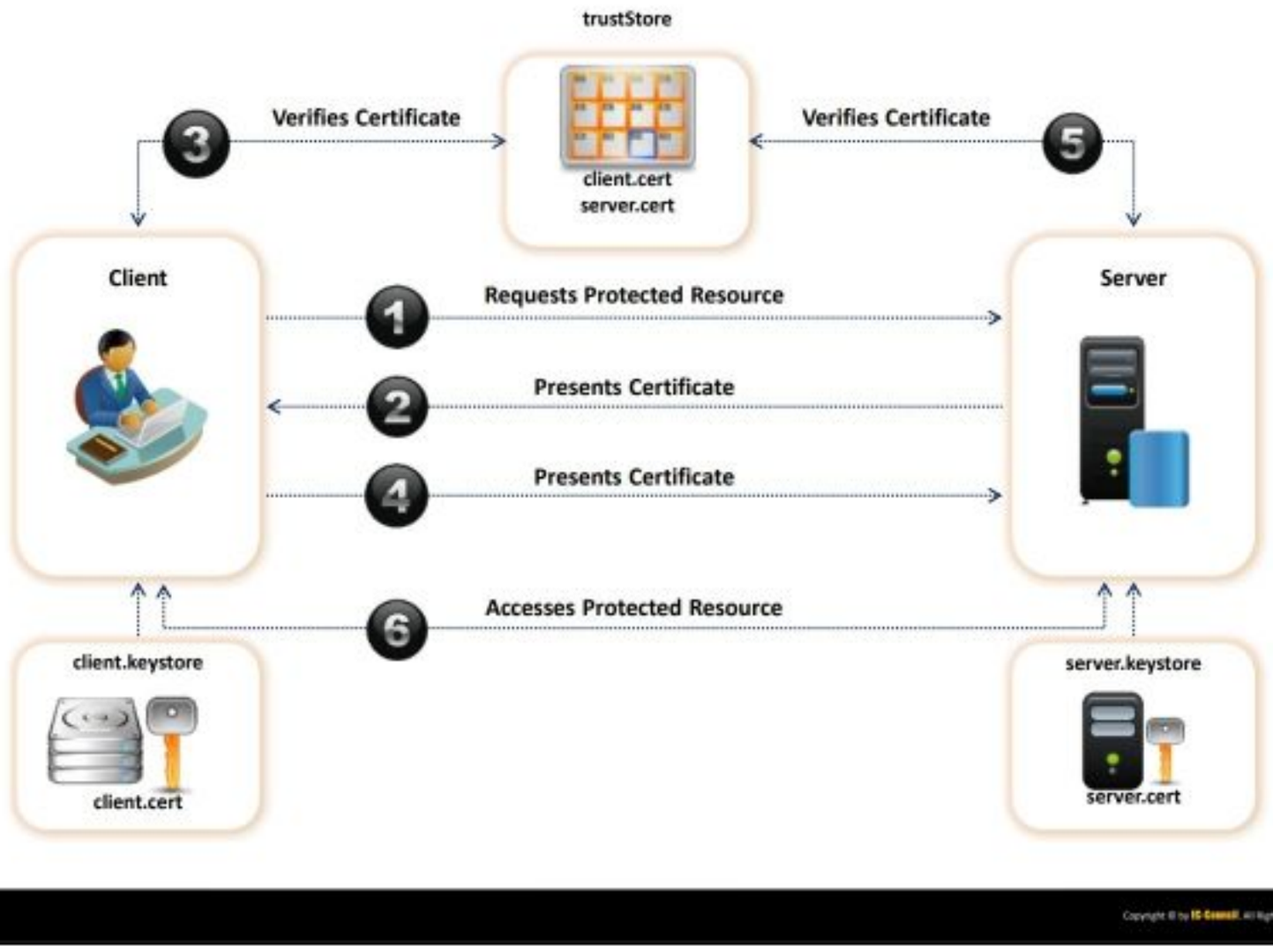
@ErrorCatchers

Client Certificate Authentication

- Client authentication **ensures** whether the users are **legitimate** or not
- In HTTPS, browsers and webservers communicate over secured SSL encrypted connection
- Servers determine a visitor's **identity** using the following methods
  - If tomcatAuthentication="**false**", then it retrieves the username and assumes all authentication is done
  - If tomcatAuthentication="**true**", CLIENT-CERT results in the value of `org.apache.catalina.authenticator.SSLAuthenticator` is input to the application
- The client and the server present each other their certificates from the keystores in authentication process
- Authentic client and server certificate copies are stored in each of the keystores for secured SSL communication

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Client Certificate Authentication (Cont'd)



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Certificate Generation with Keytool

- **Keystore** class is a memory collection of keys and certificates
- **Keytool** is a command line tool provided by Java for generating **keystores**
- Every keystore has a unique **private key** that cannot be exported but can be copied easily
- A keytool helps in **sharing public** version of the **private key** with the server

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certificate Generation with Keytool (Cont'd)

- **Generating the Client and Server Keystores**

```
1: keytool -genkeypair -alias serverkey -keyalg RSA -dname "CN=Web Server,OU=Application
Development,O=Highwinds,L=Winter Park,S=FL,C=US" -keypass password -keystore
server.jks -storepass password
2: keytool -genkeypair -alias clientkey -keyalg RSA -dname "CN=client,OU=Application
Development,O=Highwinds,L=Winter Park,S=FL,C=US" -keypass password -storepass password
-keystore client.jks
```

- **Exporting the Client's Public Certificate and Importing it in to the Server's Keystore**

```
1: keytool -exportcert -alias clientkey -file client-public.cer -keystore client.jks -
storepass password
2: keytool -importcert -keystore server.jks -alias clientcert -file client-public.cer -
storepass password -noprompt
3: # view the contents of the keystore (use -v for verbose output)
4: keytool -list -keystore server.jks -storepass password
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

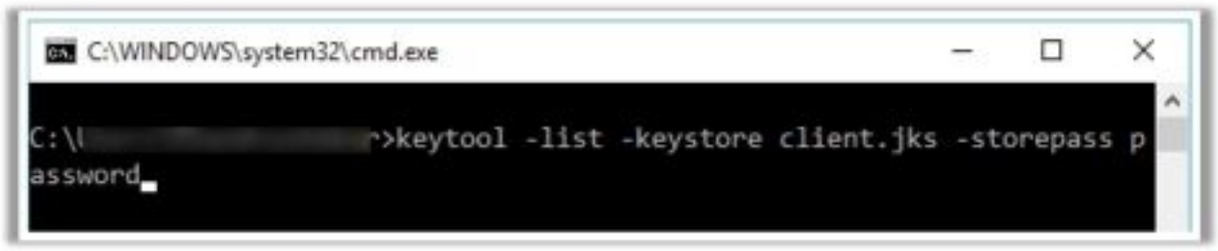
Certificate Generation with Keytool (Cont'd)

- **Export the Server's Public Certificate and import it in to the Client's `Keystore`**





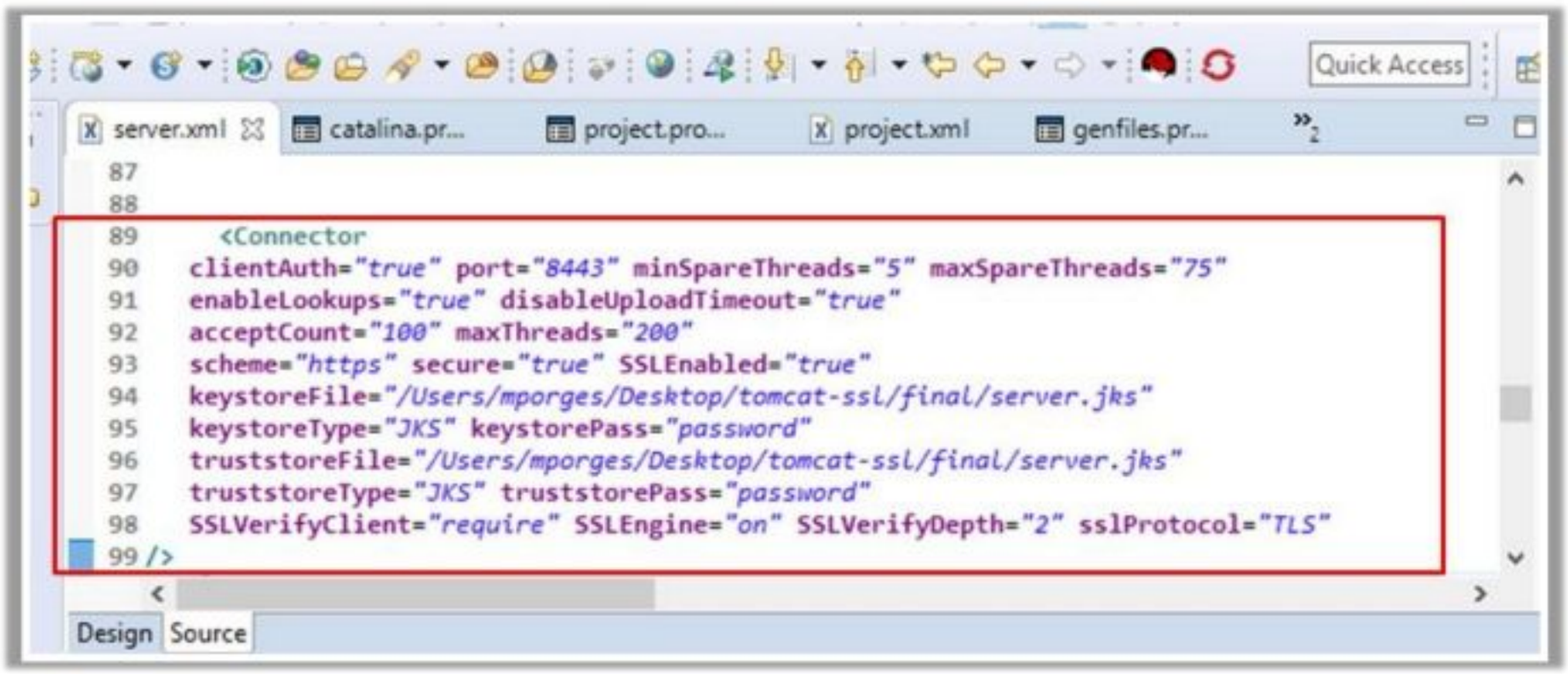

```
# view the contents of the keystore (use -v for verbose output)
```



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certificate Generation with Keytool (Cont'd)

- **Configure Tomcat for SSL using the Server `Keystore`**



Note:

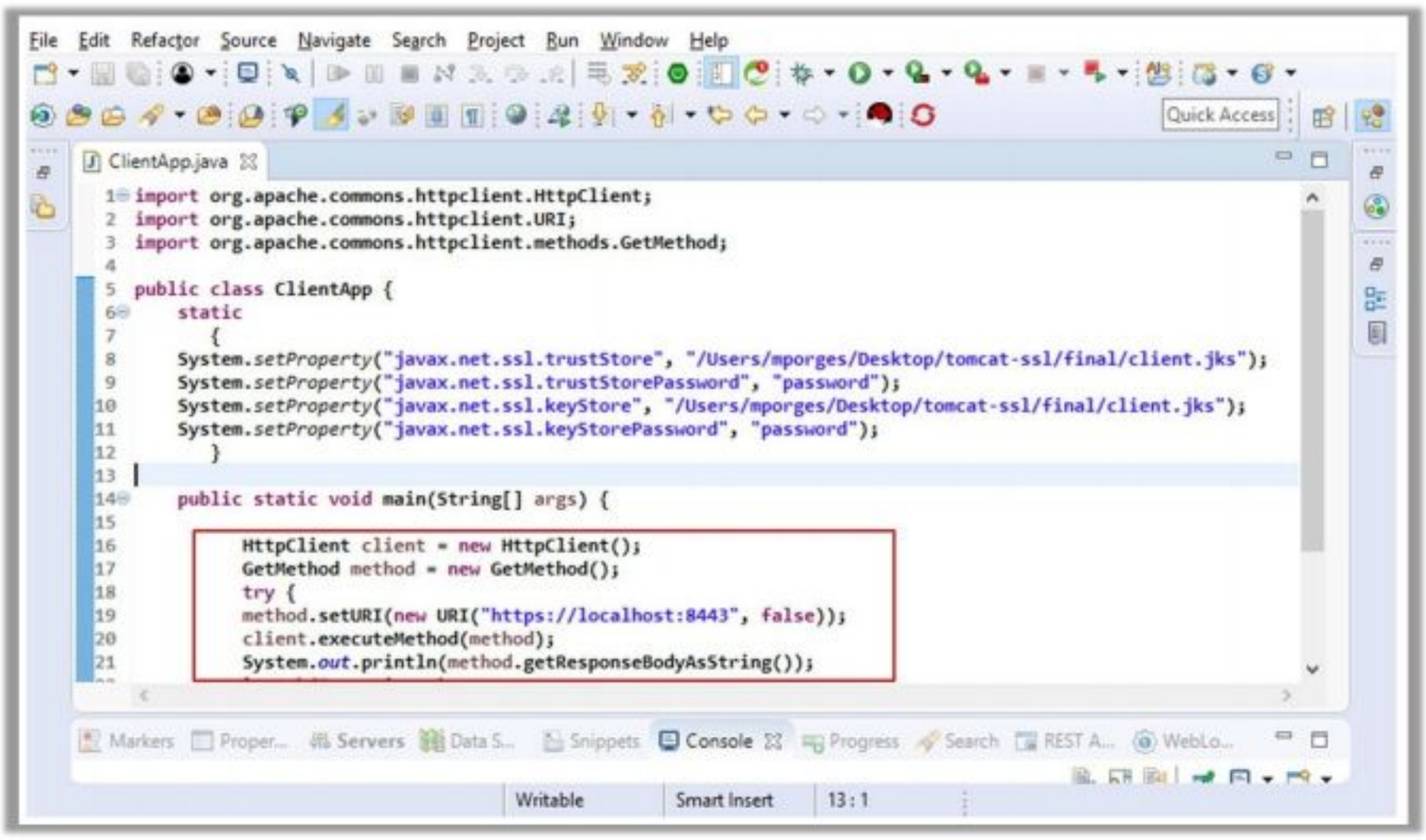
- Appropriate paths and passwords substituted for **truststorePass**, **truststoreFile**, **keystorePass**, and **keystoreFile** in `{tomcat.home}/conf/server.xml`
- **clientAuth attribute** enquires clients to provide authentication certificates

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Implementing Encryption and Certificates in Client Application



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Authentication Weaknesses and Prevention

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Authentication Weaknesses and Prevention: Brute Force Attack

- It is a continuous **trial** and **error method** of attempts to login to a web application using dictionary and other permutation combined characters

Preventive Measures

- Mandate web application accepts only increased length of **user id credentials**, especially password
- Impose web application that accepts only user id credentials that contain all valid characters including **special characters** like **!, @, #, $,** etc.
- Impose **time delay** between failed attempts
- Lock out targeted account access after a certain **restricted failed attempts**
- Impose a **password aging policy**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

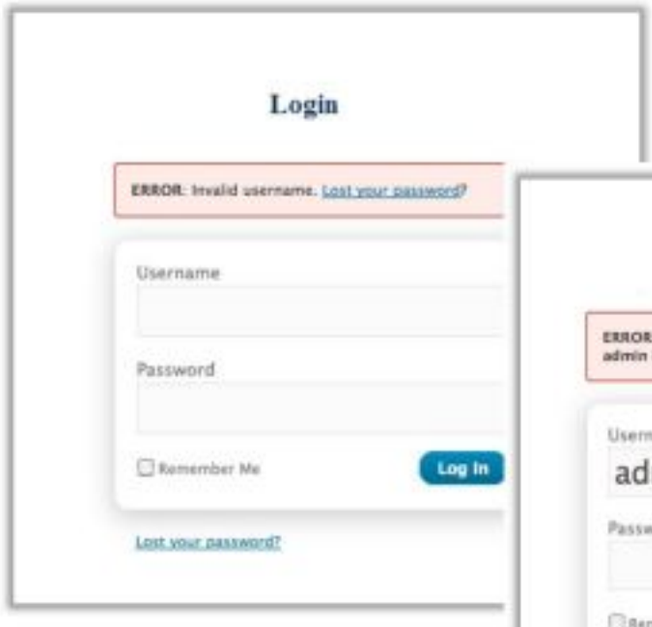
Authentication Weaknesses and Prevention: Web-based Enumeration Attack

- Attacker experiments with various **user ID credentials** and analyses the responses to find valid ones through **error message**

Web-based Enumeration Attack Prevention

- Web applications need to respond with similar error messages to all **authentication failures**
  - Credential submitted are not valid
- Analyze **URLs** and their responses during security testing to authentication failures and **prevent** unnecessary **information leakage**
  - http://www.foo.com/err.jsp?User=baduser&Error=Q
  - http://www.foo.com/err.jsp?User=gooduser&Error=2
- Analyze web page titles and their responses during **authentication failures** and prevent unnecessary information leakage
  - Invalid user
  - Invalid authentication

Indicates non existent user id



Indicates user ID is correct but password is not



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Authentication Weaknesses and Prevention: Weak Password Attacks

- Attackers **crack** weak passwords by guessing, using default user credentials, tools and techniques

Weak Password Attack Prevention

1. Mandate web application that accepts only **increased length** of user ID credentials, especially password
2. Mandate passwords to be at least **6 characters** long with a combination of uppercase and lowercase letters, digits, and special characters
3. Impose a **password aging policy**
4. **Incorrect** authentication failure messages should be avoided
5. Implement **account lockout policy**
6. Highly critical applications need **multi-factor authentication**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Introduction to Authorization

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Introduction to Authorization

- Authorization is the process that controls **access rights** of principals to system resources that include:
  - Access to users
  - Access to processes
  - Access to machines

- Main goals of authorization is to provide information security attributes, i.e., **Confidentiality**, **Integrity** and **Availability** (CIA)

- Implementation of authorization in Java is through JAAS that determines authenticated user's **purview** of given **rights**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Introduction to Authorization (Cont'd)

- It is the process of defining **access policies** for confidential resources
- Authorization shows "who is **executing**" "where is the code **residing**" "who is the **owner** of the code"

Steps to implement authorization

1. Defining **roles** to users
2. Define **security roles** of an application to roles defined in memory realm
3. Define the **resource** and **HTTP** method constraints for an application in web.xml file
4. Apply the constraints that are accessible by role
5. Check for user **authentication** for the application
6. Allow the server to **process** the **request** when the details are verified

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

JEE Based Authorization

- JEE security handles roles for **managing suitable access** to appropriate resources by **users** and **groups**
- Authorization in JEE is of two forms, **programmatic** and **declarative** authorization

Declarative

- In declarative authorization, the security information is listed in **deployment descriptors**
- These deployment descriptors are accessed by application servers and determine whether a role is assigned to a user and also decides if defined role is able to access a particular **resource**

Programmatic

- Programmatic authorization is performed by using **APIs** and **functions** such as `isUserInRole()`

Example code for Declarative Authorization

LoginServlet.java | web.xml

```
16
17      <auth-constraint>
18  <description>Must authenticate before querying the
19          <role-name>Admin</role-name>
20      </auth-constraint>
21      <user-data-constraint>
```

Example code for Programmatic Authorization

LoginServlet.java

```
25          if(req.isUserInRole("Admin"))
26          {
27              System.out.println("welcome");
28          }
29
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

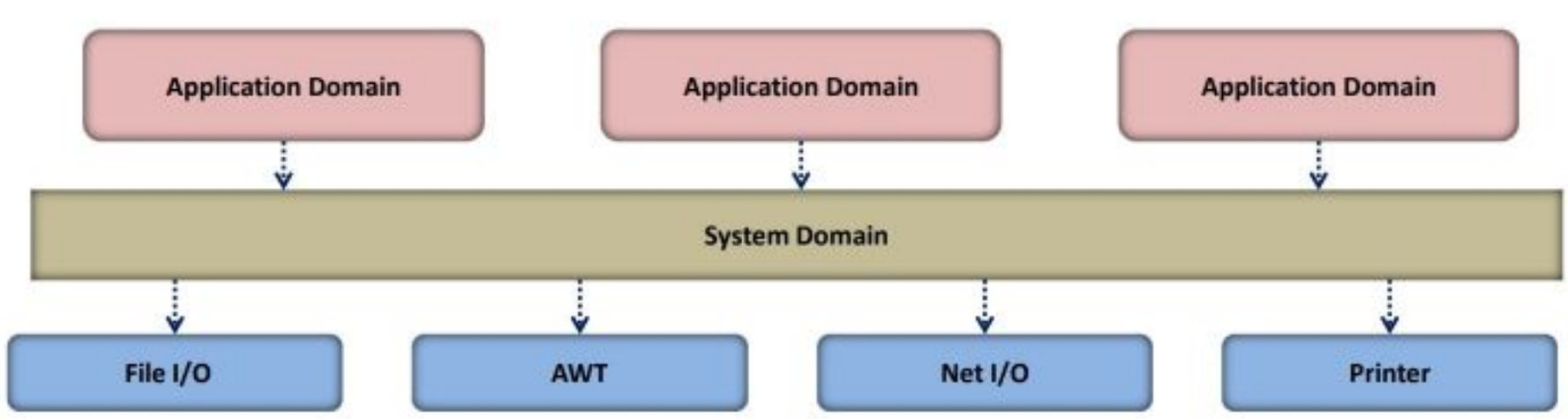
Access Control Model

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Access Control Model

- All the protected resources in Java applications are accessed only through **system domain**
- Protected resources include keyboard, AWT, printing systems, networking facilities and file systems
- Permissions to running programs are checked by calling the method `SecurityManager.checkPermission()`, ensuring proper **permissions** are given to the objects
- While checking permissions, `SecurityManager.checkPermission()` invokes `AccessController.checkPermission()` to **verify** the permissions



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Access Control Model (Cont'd)

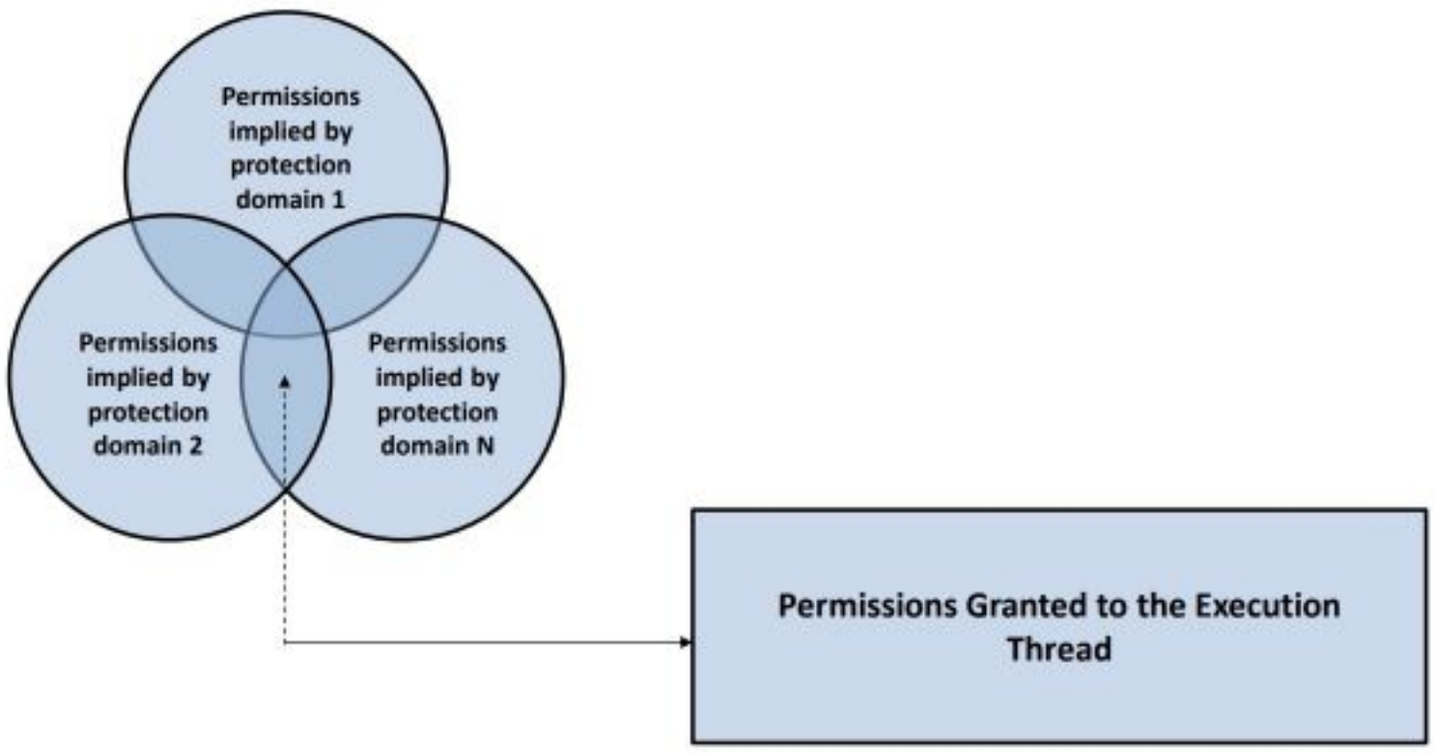
- The **`AccessController.checkPermission()`** method logic ensures each class in the thread stack gets `ProtectionDomain`
- All these `ProtectionDomains` are composed into a `java.security.AccessControlContext` object
- **`implies()`** method is invoked along with permission object on the `ProtectionDomain` to check as a parameter
- In case if checked permissions are invoked by `ProtectionDomain` for each stack frame, this continues even for the next frame on the stack
- Testing continues until stack ends, concluding that all the thread's classes have proper permission for the operation's success

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Access Control Model (Cont'd)

- Even if one of the `ProtectionDomain` in the thread's stack does not have **checkPermission**, then access to resource is denied along with a `SecurityException` thrown



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Access Control Model (Cont'd)

- Both authentication and authorization play an important role in **information security** and together form access control

Access control comprises three models:

1. **Discretionary** access control (DAC)
2. **Mandatory** access control (MAC)
3. **Role – based** access control (RAC)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Discretionary Access Control (DAC)

- This works based on the **notion of ownership**
- If an individual creates a file then he/she becomes the owner of the file and identified as **data owner**
- For example, Bob creates a file called student.txt. He wants Alice to read the file so he provides access to read the file but do nothing else such as **editing** or **deleting**
- In this scenario, Bob is the data owner and Alice is the user who is **authorized** to view the file created by Bob
- DAC is prescribed through **Access Control Lists** (ACL's) that stores rules of access to resources
- ACLs contain subjects (users) and objects (resources)

- The subjects are allowed to **access objects** by defining a set of privileges
- DAC can **grant** and **revoke** the rights to resources
- **Capability list** stores a list of rights an object has for each subject called capabilities
- Authentication happens when **capabilities** are **granted**
- Limitation of DAC is that it is very **hard** and **complex** to implement in big enterprise scenarios
- It is **tedious** to define rights or privileges per individual as it leads to large access control rules for each file

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Mandatory Access Control (MAC)

- This model relies on the principle of **limiting information dissemination**, and policies imposed by every data item in the system are governed by **Central Authority** throughout lifetime
- The creator of object (file) does not necessarily know who has **authorized access** to it
- The access control policies are imposed on policy, **specific** to the **user**
- As opposed to discretionary access control (DAC), there is no concept of data owner, **confidentiality** is a sign of access control that guards the ability to read data
- Policies may vary depending on each action being performed on the resources, i.e., functions for a file in a file system are **read**, **write**, **delete**, **create**, etc.

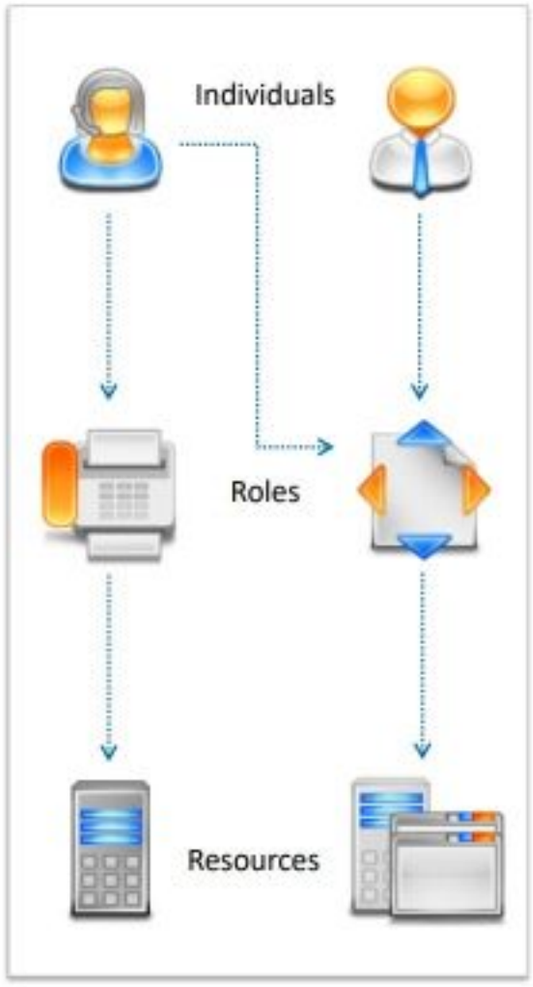
- Objects (file which contains data) in the system are assigned **clearance levels** that are specified as unclassified, classified, secret and top secret
- A user with clearance level of classified is allowed to read **unclassified information** but not secret and top secret information
- Some examples of MAC are **official** reports, **accounting** reports, **medical reports statistical analysis**, **e-business applications**, etc.
- Access control must be mandatory whenever possible from system end user's perspective, and it protects the system from **operational security errors** and also makes the system simpler

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Role-based Access Control (RBAC)

- This is the **popular** access control model
- A user has access to resources based on the **role assigned**; roles are allocated depending on job function
- Permissions are distinct on job authority and responsibilities contained in job function. They also determine **interaction** between subject and object
- It functions on the concept of user **roles** and **information** accessibility
- An organization has different departments, and roles are assigned based on **requirements**; RBAC provides only access and privileges needed by an individual
- For **example**, the Human Resource Department does not have a need to view account department's information; the role of HRD is to view human resources information



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Servlet Container

1. A component of web server that facilitates **loading**, **executing** and **initializing** servlets in standalone (without a web server) or by running on another host
2. Examples of servlet container are **tomcat**, **Glassfish**, etc.
3. A web server uses Hyper Text Transfer Protocol (HTTP) to **communicate** with end users, which are generally web browsers
4. A Java-based web server communicates through HTTP messages by using two main classes, **java.net.Socket** and **java.net.ServerSocket**
5. Servlet container **manages** large number of **servlets** ; hence, it can handle many requests
6. For each Http method (`Get`, `Post`), **HttpServlet** class has a corresponding method (**`doGet()`**, **`doPost()`**, **`doPut()`**)
7. Servlet container calls the **`service()` method** of servlet and provides an instance of ServletRequest and ServletResponse that enables in identifying remote user, parameters of HTTP `POST`

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Authorizing Users by Servlets

- To permit access to authorized users, servlets can be configured by:

Defining Roles

- An application function can be defined by **security** roles that are made up of users, groups or both users and groups
- The relationship between users and groups is identified based on the implementation used
- Roles can be defined in Java EE deployment descriptor file as **web.xml** and corresponding mapping to roles in the web server deployment descriptor file as **serv-web.xml**

Defining Constraints

- Access permissions can be defined using auth-constraint element in **web.xml** file
- Web containers impose authorization at **page level**
- Once the user is authenticated, the auth-constraint defined in the **deployment descriptor** checks if the user belongs to any one of the roles

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Authorizing Users by Servlets (Cont'd)

Example for defining roles

- Each element of **security-role-mapping** in **web.xml** file maps role names to principals and groups that are allowed by web applications
- The role-name must match with role-name of security-role element of the equivalent **web.xml** file

Example of Defining Constraints for Servlet Authorization Servlets

LoginServlet.java | web.xml

```
43  <security-constraint>
44<web-resource-collection>
45<web-resource-name>LoginPages</web-resource-name>
46<description> access  authorized users only </description>
47<url-pattern>/manager/*</url-pattern>
48<http-method>GET</http-method>
49</web-resource-collection>
50<auth-constraint>
51<description>The roles who can access are</description>
52<role-name>clerk</role-name>
53</auth-constraint>
54</security-constraint>
55
56   <security-role-mapping>
57 <role-name>clerk</role-name>
58 <principal-name>john</principal-name>
59 <principal-name>mweb</principal-name>
60 <group-name>HR</group-name>
61 </security-role-mapping>
62 <security-role-mapping>
63 <role-name>manager</role-name>
64 <principal-name>Sinha</principal-name>
65 </security-role-mapping>
66</web-app>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

EJB Authorization

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

EJB Authorization Controls

- To secure Enterprise JavaBeans (EJBs), J2EE architecture security model is supported by **WebLogic Server**
- Securing the EJB authorization includes **declarative** authorization and **programmatic** authorization
- J2EE architecture has a **container** that serves as an authorization boundary between the components, its hosts and their callers
- Authorization container resides within the container's **authentication boundary** in order to implement the secure authentication process
- Container allows the call only after it validates the **credentials** of the callers along with the control rules (capabilities and permissions) for the target component

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Declarative Security with EJBs

1. EJB or the Enterprise JavaBeans Technology is Java EE's server-side component architecture. It helps in developing **portable**, **secure**, **traditional** and **distributed** applications
2. Deployment descriptors such as **ejb-jar.xml** and **weblogic-ejb-jar.xml** are used in the EJBs to implement declarative security
3. These descriptors are used for the purpose of **defining security** specifications
4. The EJB container uses such **security definitions** at runtime to enforce the requirements
5. Descriptors also map the application's security specifications with its **runtime definitions**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Declarative Security with EJBs (Cont'd)

- **Following are the steps to configure security in the EJB deployment descriptors:**

1. Use a text editor to create **ejb-jar.xml** and **weblogic-ejb-jar.xml** deployment descriptor files
2. Define constraints for security role name, EJB name and the method name in **ejb-jar.xml** file
3. Security role names are **case sensitive**; follow some restrictions and conventions when specifying role names
4. In **weblogic-ejb-jar.xml** descriptor file, define security role name and associate it to one or multiple principals (users or groups) in a security realm

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Declarative Security with EJBs (Cont'd)

- **The following code shows the mapping of security role names to a security realm by using ejb-jar.xml and weblogic-ejb-jar.xml files**

ejb-jar.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <assembly-descriptor>
3 <security-role>
4 <role-name>clerk</role-name>
5 </security-role>
6 <security-role>
7 <role-name>bob</role-name>
8 </security-role>
9 <method-permission>
10 <role-name>clerk</role-name>
11 <role-name>bob</role-name>
12 <method>
13 <ejb-name>accountsPayable</ejb-name>
14 <method-name>getReceipts</method-name>
15
16 </method>
17 </method-permission>
18 <!--  ...-->
19 </assembly-descriptor>
20 <!-- ...
21 weblogic-ejb-jar.xml entries:  --><security-role-assignment>
22 <role-name>clerk</role-name>
23 <principal-name>ec</principal-name>
24 <principal-name>John</principal-name>
25 <principal-name>Jacob</principal-name>
26 </security-role-assignment>
```

Design | Source

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Programmatic Security with EJBs

- **javax.EJBContext.getCallerprincipal()and javax.ejb.EJBContext.isCallerInRole() methods can be used to implement programmatic security in EJBs**

getCallerPrincipal

- By using the method `getCallerPrincipal()`, the EJB caller can be determined
- If one exists in the `subject` of calling user, WLSUser Principal is returned by method `javax.ejb.EJBContext.getCallerPrincipal()`
- If there are multiple WLSUser Principals, the first one in `Subject.getPrincipals().iterator()` method is returned
- In case there are no WLSUser Principals, the first non-WLSGroup Principal will be returned by method `getCallerPrincipal()`
- If principals are not present or all principals belong to WLSGroup, the method returns `weblogic.security.WLSPrincipals.getAnonymousUserPrincipal()`

isCallerInRole

- This method determines whether the caller is assigned a security role and authorized to perform actions on the resource of WebLogic in the executing thread

Example, `javax.ejb.EJBContext.isCallerInRole("admin")` returns true if the current user has admin privileges

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Java Authentication and Authorization Service (JAAS)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java Authentication and Authorization (JAAS)

- Java Authentication and Authorization Service is used for both **authentication** and **authorization** of users
- For every Java (J2EE) application, **authentication** and **access control** are considered as important security measures
- Using JAAS, a security standard can be implemented for all **Java applications**
- JAAS provides a **pluggable** and **flexible framework** that enables developers to implement various security mechanisms and security sources
- Authentication in JAAS determines the user who is currently **executing** the Java code, regardless of the environment (an application, servlet, applet, etc.)
- Authorization in JAAS determines the **control rights** of the user that is required to perform specified actions
- **Pluggable Authentication Modules** (PAM) are implemented in JASS for the purpose of authentication in JAAS
- JAAS was integrated in **J2EE** (Java 2 Enterprise Edition) and **JDK 1.4**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

JAAS Features

- JAAS is implemented using pure **Java**
- Authentication of users is done through **PAM** framework
- JAAS supports single **sign-on** for login authentication in J2EE applications
- JAAS provides centralized role-based control that includes **hierarchical** roles
- JAAS provides access control **policies** for user-based, group-based, and role-based authorization

Authentication



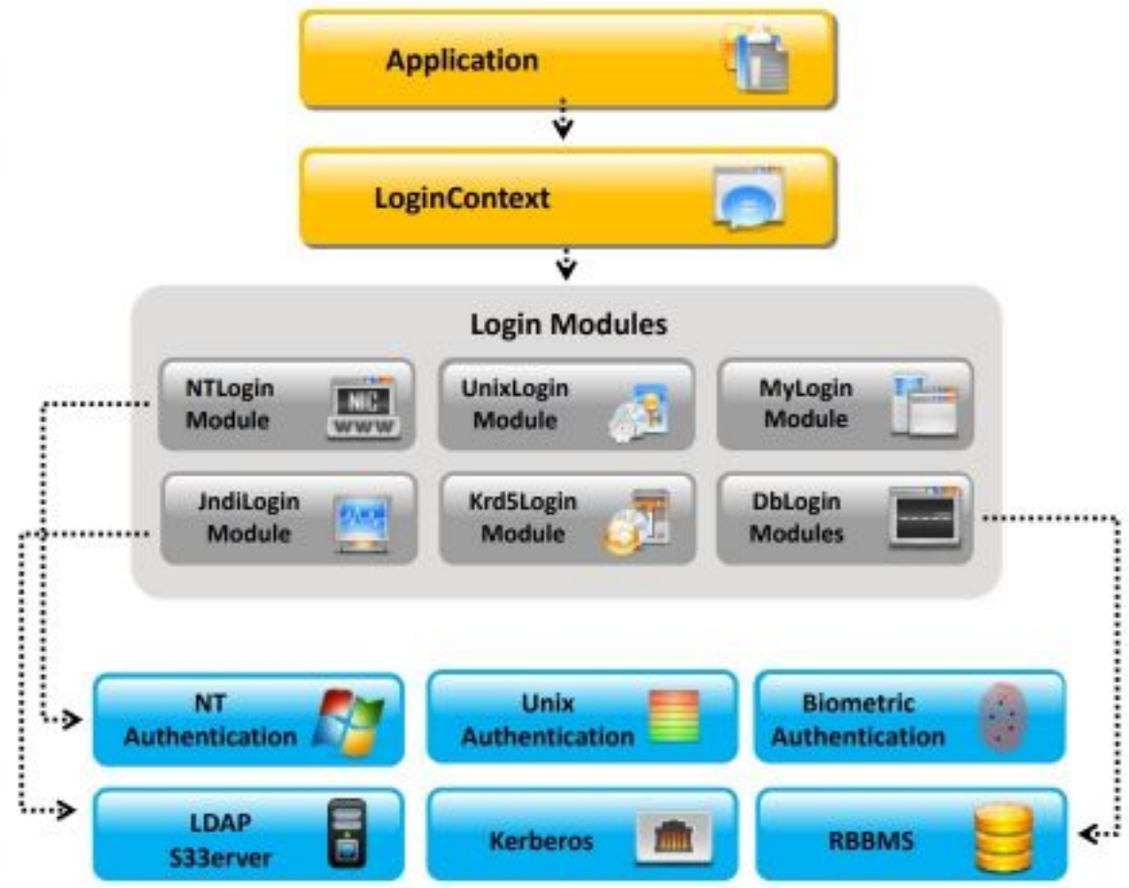
Authorization



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

JAAS Architecture

- The application layer initially deals with **LoginContext**
- Next to LoginContext are one or more **LoginModules** that are dynamically configured
- These LoginModules make use of appropriate security infrastructure to achieve authentication
- JAAS also provides reference LoginModule implementations such as **JndiLoginModule**
- JAAS is **stackable** in addition to being pluggable
- Single login implementation works through ordered security module stacks, featured one over the other, interacting with the Java architecture



A High-level Overview Of JAAS

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Pluggable Authentication Module (PAM) Framework

1. The **authentication** framework of JAAS is based on PAM
2. PAM enables authentication in a pluggable fashion that supports the addition of more authentication **modules** to the application
3. Java application functions independently regardless of the implemented authentication **technologies**
4. New authentication technologies can be implemented without the need of **modifying** the application
5. PAM enables different **web applications** to authenticate users against its own data store
6. PAM is the feature that helps programmers to implement single sign-on in web applications that are deployed on the same **application server**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

JAAS Classes

JAAS Common Classes

- **Subject** (`javax.security.auth.Subject`)
- **Principal** (`java.security.Principal`)
- **Credential**

JAAS Authentication Classes

- **Callback** (`javax.security.auth.callback.Callback`)
- **CallbackHandler** (`javax.security.auth.callback.CallbackHandler`)
- **Configuration** (`javax.security.auth.login.Configuration`)
- **LoginContext** (`javax.security.auth.login.LoginContext`)
- **LoginModule** (`javax.security.auth.spi.LoginModule`)

JAAS Authorization Classes

- **Policy** (`java.security.Policy`)
- **AuthPermission** (`javax.security.auth.AuthPermission`)
- **PrivateCredentialPermission** (`javax.security.auth.PrivateCredentialPermission`)

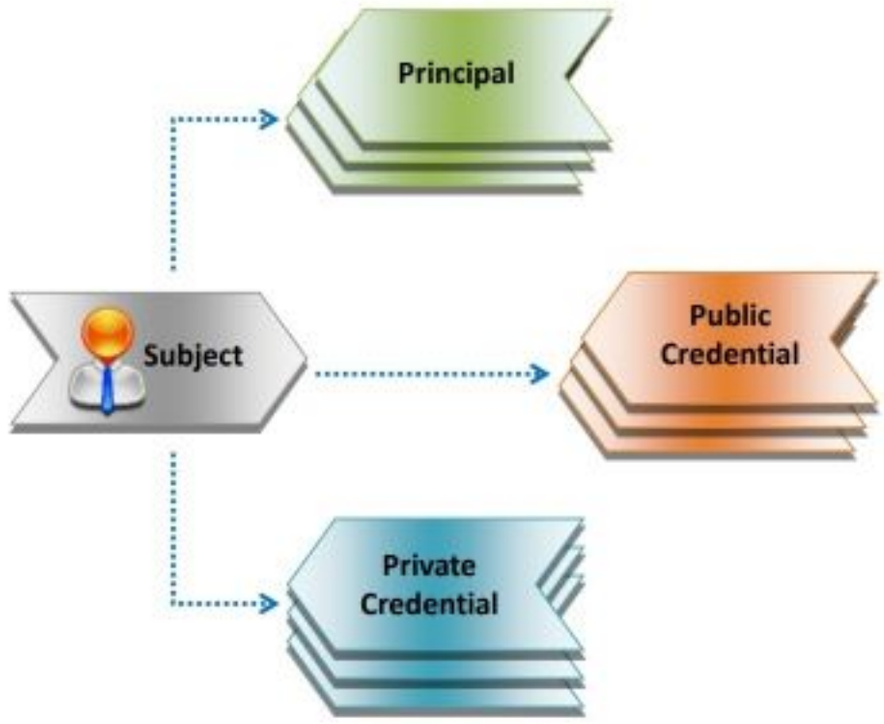
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

JAAS Subject and Principal

- The Subject class is considered as the **central class** of JAAS
- The Subject represents information for a single **user**, **entity** or **system**
- The Subject includes the entity **principals**, **public** (public keys) and **private** (passwords, private keys) credentials
- A Principal **encapsulates** features or properties of a subject
- The JAAS API interface used to represent Principal is `java.security.Principal`
- A Subject is associated with **identities** or principals throughout the authentication process
- A Subject can contain **multiple principals**

For Example, a user with a name principal (George), a social security number principal (123-12-1234) and a username principal (GeorgeD), all of which differentiates the subject from other subjects



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

JAAS Subject and Principal (Cont'd)

- **To retrieve principals from the associated subject, the following methods can be used**

Methods	Description
`public Set getPrincipals() {...}`	Retrieves all the Principals from the specified subject
`public Set getPrincipals(Class c) {...}`	Retrieves only the Principals that are instances of class 'c' or its subclasses
`Pubic Set getPrivateCredentials() {...}`	Retrieves the private credentials from the specified subject
`Public Set getPrivateCredentials (Class c) {...}`	Retrieves only the private credentials that are instances of class 'c' or its subclasses
`Public Set getPublicCredentials () {...}`	Retrieves the public credentials from the specified subject
`Public Set getPublicCredentials (Class c) {...}`	Retrieves only the public credentials that are instances of class 'c' or its subclasses
`Public boolean isReadOnly() {...}`	Tests a Subject's read only state
`Public void setReadOnly() {...}`	To set Subject to read only state

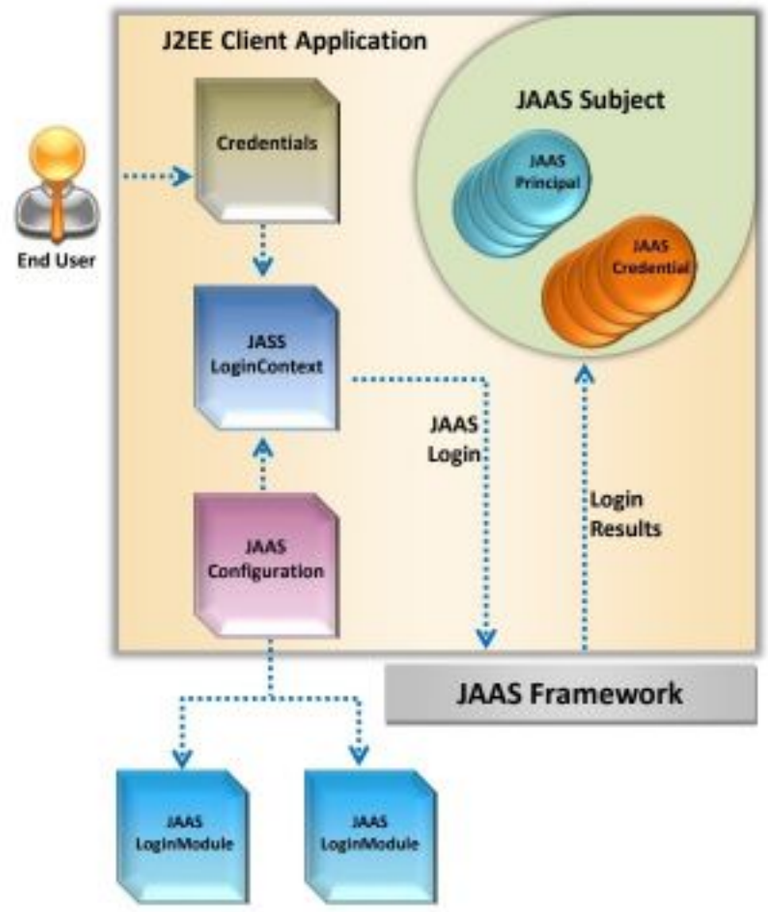
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Authentication in JAAS

- Authentication is used to verify whether the user is **eligible** to **access** the application
- JAAS enables **plugging** of various authentication modules at **runtime**
- The client application interacts with JAAS through the **LoginContext object**
- Authentication in JAAS can be implemented using the following set of modules as represented in the diagram
  - LoginContext
  - LoginModule
  - CallbackHandler



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Authentication Steps in JAAS

The initial step is to create a **LoginContext object**. The LoginContext object checks with the configuration file to identify which **LoginModule** is to be loaded. Optionally, a CallbackHandler can be passed to the LoginContext

Login method from LoginContext is called to implement authentication. This process loads the predefined LoginModule to verify if the user can be authenticated

Integrate credentials and principals with the subject for successful logins

In case of login failure, throw a **LoginException**

To log out from application, make use of **logout** function from LoginContext

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Authorization in JAAS

- Authorization of an user is used to identify whether the (authenticated) user is **eligible** to **perform** certain **functionalities** in the application like accessing a resource
- Since JAAS is developed on the existing Java security model, the process of authorization is **policy-based**
- The policy configuration file consists of a list of entries like "**keystore**" and "**grant**"
- The grant entry provides with the permissions granted for the authenticated codes or principals to perform the security-sensitive operations
- JAAS also supports **principal-based** policy entry
- Permissions are granted in policy with regard to **specific principals**

The basic format of a grant entry looks as:

```
1: grant Codebase "codebase_URL" Signedby
   "signer_name,"
2: Principal principal_class_name
   "principal_name",
3: Principal principal_class_name
   "principal_name",
4: S {
5: permission permission_class_name
   "target_name", "action",
6: permission permission_class_name
   "target_name", "action",
7: S
8: }
```

Source: *http://www2.sys-con.com*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Authorization in JAAS (Cont'd)

- The **Policy object** in JAAS indicates the system security policy for a Java application environment
- Once the authentication of user is done, the authorization takes place through the `Subject.doAs` function or `doAsPrivileged` function from Subject class
- The `doAs` function dynamically combines the Subject with the current **AccessControlContext** and then triggers the run method to execute the security checks

Access Manager Policy Framework

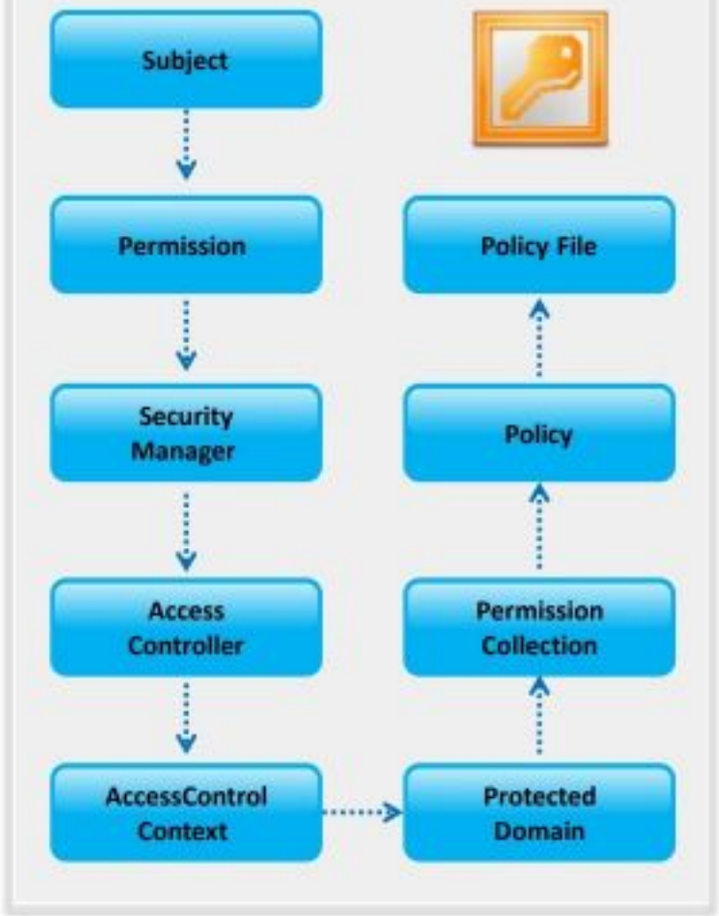
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Authorization Steps in JAAS

1. The initial step is to invoke `Subject.doAs` or `doAsPrivileged`
2. Check the permission by calling `SecurityManager.checkPermission` or other check methods
3. The SecurityManager redirects the check to the **AccessController**
4. The AccessController checks whether the relevant AccesControlContext includes all the **permissions** required for the action to be performed
5. AccessControlContext is updated by the **SecurityManager** with the permission policies given to the Subject



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

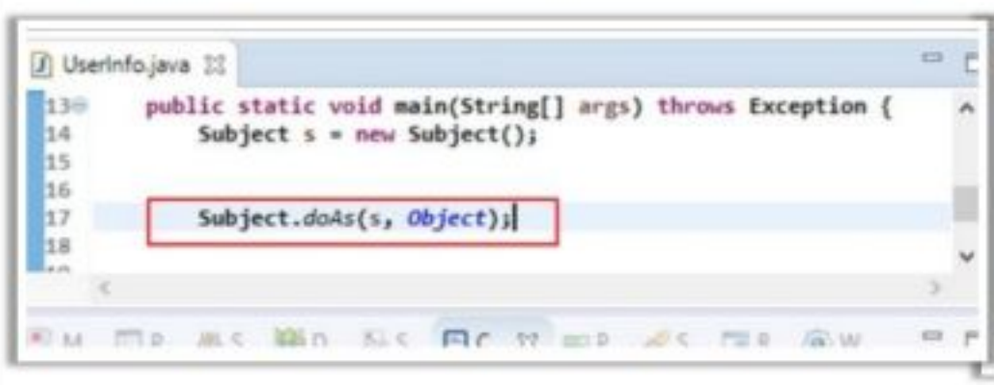
Subject Methods doAs() and doAsPrivileged()

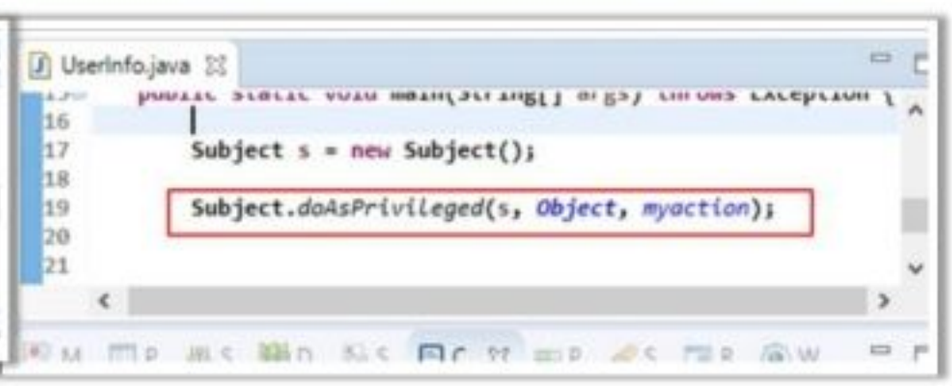
Object doAs (Subject, PrivilegedAction)

- The specified privileged action is carried out as the **specified Subject**
- This method combines the Subject with the AccessControlContext instance of the current thread adding the Subject's permissions to the permissions of that access control context
- A new access control context is created with the combined permissions

Object doAsPriviledgedAction (Subject, PrivilededAction, AccessControlContext)

- This method also possesses the same functionality as the `doAs ()` method but within the specified access control context
- It is mainly used for specifying a null access control context





Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Impersonation in JAAS

- JAAS can be used to implement impersonation at the **application level**
- The Subject class in JAAS enables the execution of a specific **sensitive operation** as another user's entity

```
Subject.java
3  public class Subject {
4      //...
5      //...
6      //associate the subject with the current AccessControlContext and execute the P
7      public static Object doAs(Subject sub, java.security.PrivilegedAction paction )
8      {
```

- When the above code is executed, the **doAs()** method links the impersonated Subject with the current AccessControlContext and performs the action
- The Subject is removed from the AccessControlContext at the end of **doAs method call**

```
16 {
17     public static void main(String args[])
18     {
19
20         Subject sub=loginContext.getSubject();
21         Subject.doAs(sub, new ProtectedOperation());
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

JAAS Permissions

- Permissions play an important role in authorization
- Permissions control the access to **resources**
- JAAS permissions are developed from the existing Java security model
- This model provides an efficient way of controlling access to resources like **sockets** and **files** which excludes URLs
- To integrate JAAS with a web application, a new **permission class** should be created
- The permission class can be created using the following two ways:
  - To extend `java.security.BasicPermission`, which links the permissions to URLs
  - To create URLPermission class that extends `java.security.Permission` class

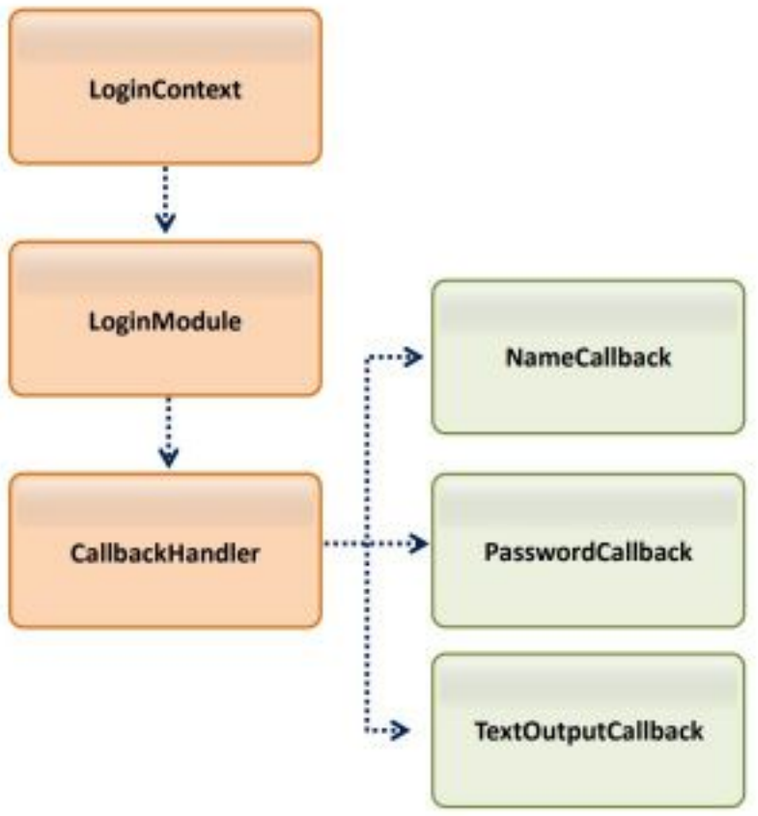
```
Permission.java
1  package java.security;
2
3  import java.security.PermissionCollection;
4
5  public abstract class Permission {
6  public Permission (String sname) {}
7  public boolean implies (Permission permsn) {
8      return false;}
9  public boolean equals (Object ob) {
10     return false;}
11 public String toString() {
12     return null;}
13
14 public PermissionCollection newPermissionCollection() {
15
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

LoginContext in JAAS

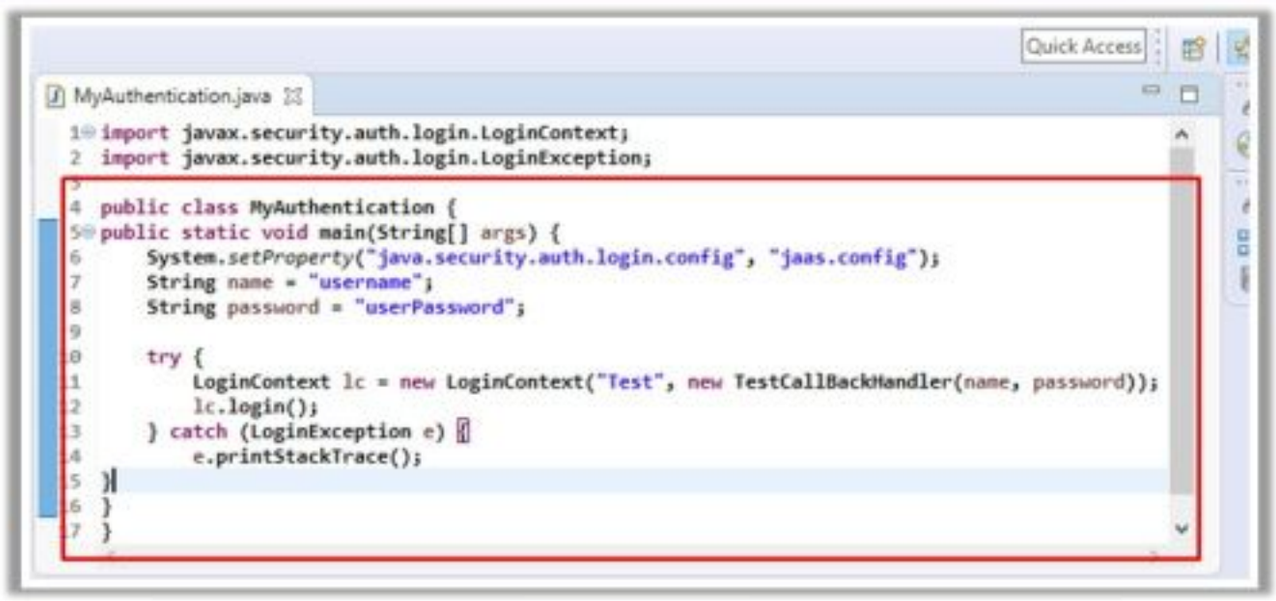
- The **LoginContext** is mainly used for the log in purpose
- The code used for authentication creates a new instance of a LoginContext, and it in turn uses a **Configuration** to identify which login module is used to authenticate a subject
- The login code then executes the login() method from the **LoginContext**



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Creating LoginContext

1. The **name parameter** is used to get the login Configuration
2. The constructor without a **`CallbackHandler`** class depends on the default `CallbackHandler` or on the `LoginModules` to get information
3. Callers of a `LoginContext` constructor should require an AuthPermission with target **"createLoginContext.<name>"**



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

LoginContext Instantiation

- `LoginContext` uses **application identifier** to load the appropriate configuration block

MyAuthentication.java

```
15
16
17  LoginContext loc=new LoginContext("RdbmsLoginModule");
18  loc.login();
19
20
```

RdbmsLoginModule.java

```
27  @Override
28  public boolean login() throws LoginException {
29      //code for login condition
30      return false;
31  }
```

- LoginContext locates the JAAS configuration file by the **`javax.security.auth.login.config`** system property as shown here:

  "java -Djavax.security.auth.login.config=jaas.config MyApp"

JAAS Configuration

The configuration file format consists of the following entries:

- **LoginEntry**: Labels a single entry in the login configuration. The application code implicitly defines the LoginEntry label where a particular **LoginEntry** entry is searched by its login configuration
- **ModuleClass**: Specifies the class name of a JAAS login module
  Example: **`com.sun.security.auth.module.Krb5LoginModule`** specifies the class name of the Kerberos login module
- **Flag** : Specifies the way to respond when the current login module gives an authentication failure. The flag can be set with anyone of the following values
  - **Required** - Authentication should succeed irrespective of success or failure, proceed to the next login module in this entry
  - **Requisite** - Authentication should succeed. For success condition, proceed to the next login module, and in case of failure, return immediately without executing the remaining login modules
  - **Sufficient** - It is not essential for this login module to succeed in authentication. For success condition, return immediately without executing the remaining login modules, and in case of failure, proceed to the next login module
  - **Optional** - It is not essential for this login module to succeed in authentication. Irrespective of success or failure, always proceed to the next login module in this entry
- **Option="Value"** - Zero or more option settings can be passed to the login module after the flag. The options are given in the form of space-separated lists. The login module lines should be terminated using a semicolon (;)

The JAAS Login Configuration File's General Format:

MyAuthentication.java | jaas.config

```
1 /* JAAS Login Configuration */
2 LoginEntry {
3     default.ModuleClass required="require" debug="true" ;
4
```


Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Locating JAAS Configuration File

The location of JAAS login configuration files follows two approaches:

1. Set a System Property

The system property value `java.security.auth.login.config` is set to the location of the login configuration file

E.g. the system property can be set on the command line as:

```
1: java -
2: Djava.security.auth.login.config=/var/activemq/config/login.config
3:  ...
```

2. Configure the JDK

JAAS checks the `$JAVA_HOME/jre/lib/security/java.security` security properties file for entries of the form if the system property is not set

```
1: login.config.url.1=file:C:/activemq/config/login.config
```

- When more than one entry is found, `login.config.url.n`, the entries should be consecutively numbered
- A single configuration file is used for storing login files in `java.security` package

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

JAAS CallbackHandler and Callbacks

- When `CallbackHandlers` are specified in application, the **`LoginModules`** remains independent regardless of the application interaction with users
- Login modules make use of **`javax.security.auth.callback.CallbackHandler`** to retrieve authentication information from the user
- The CallbackHandler **interface** is implemented in applications and then passes it to the `LoginContext` that sends it directly to the `LoginModules`
- The LoginModule makes use of the `CallbackHandler`:
  - To get input from the users

    E.g. to get password, pin number, etc.
  - To give information to users

    E.g. status information
- For **example**:
  - `CallbackHandler` implementation for a GUI application might display a window and request user for input
  - `CallbackHandler` implementation for a non-GUI might simply requests user to give input on the command line

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

JAAS CallbackHandler and Callbacks (Cont'd)

- CallbackHandler interface should implement the following method:

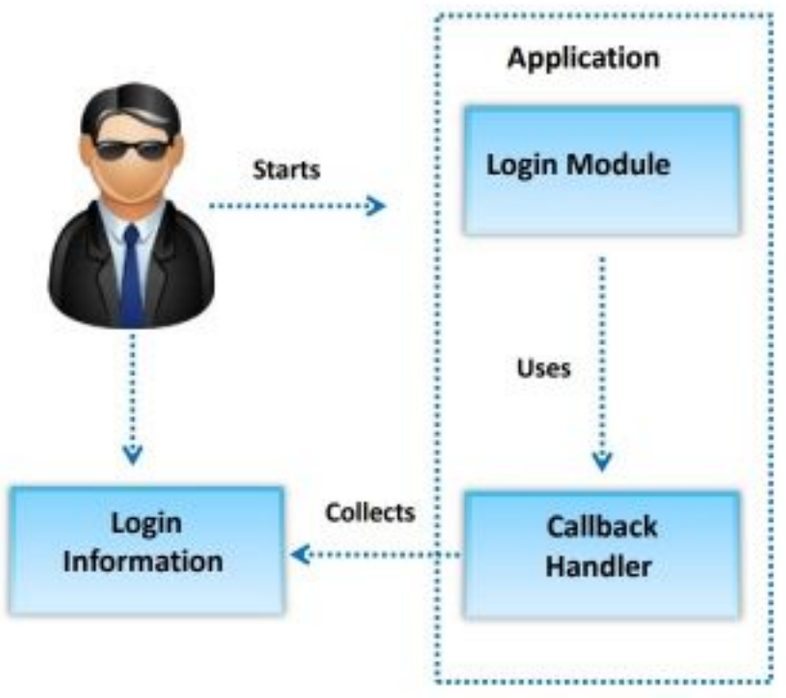
```
Quick Access
TestCallBackHandler.java
12
13  @Override
14  public void handle(Callback[] arg0) throws IOException, UnsupportedCallbackException
15  {
16
17  }
```

- To process **NameCallback**, the `CallbackHandler` initially retrieves the name from user by requesting the user and then calls the **setName** function of the `NameCallback` to store the name
- The login module passes an **array** of **Callbacks** to the `CallbackHandler` handle method to execute the requested user interaction and sets values to the Callbacks
- The package **javax.security.auth.callback** consists of the Callback interfaces and many other implementations
- Example of Callbacks include `NameCallback` (for user name) and `PasswordCallback` (for password)
- LoginModule sends an array of `Callbacks` directly to the `CallbackHandler's` handle method

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Login to Standalone Application

- **Step 1:** The user **initiates** the application
- **Step 2:** **Login** module is invoked
- **Step 3 & 4:** The Login module makes use of the **callback handler** to collect login information from the user using callbacks
- **Step 5:** Login **information** entered by the user
- **Step 6:** Login module gets the information and decides whether to allow the user into the application or not



Login to Standalone Application

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

JAAS Client

- The code below given is an example of JAAS client where :
  - The **username**, **password**, and **URL** are read as input arguments from the command line
  - Connection to the URL is made, and the client is **authenticated** using the given username and password
  - A **privileged action** is executed under the newly implemented authenticated subject

MyClient.java

```
1  import javax.security.auth.Subject;
2  import javax.security.auth.login.LoginContext;
3
4  public class MyClient {
5  public static void main(String[] args) {
6      String user = args[0];
7      String pass = args[1];
8      String url = args[2];
9      LoginContext lc = null;
10
11     try {
12
13         lc=new LoginContext("Test", new TestCallBackHandler(user, pass,url));
14
15     } catch (Exception e) {
16         e.printStackTrace();
17     }
18     try {
19         lc.login();
20
21     } catch (Exception e) {
22         e.printStackTrace();
23         System.exit(-1);
24     }
25
26     Subject sub=lc.getSubject();
27     SimpleAction simpleAction = new SimpleAction(url);
28
29 }
30 }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

LoginModule Implementation in JAAS

- The LoginModule interface should be implemented to perform authentication
- More than one login module can be used at a time and JASS logs in through each of them
- Depending upon various successful attempts, JAAS can be made to allow or deny logins

LoginModule implements 5 methods in its interface namely:

- `Initialize()`
- `Login()`
- `Commit()`
- `Abort()`
- `Logout()`

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

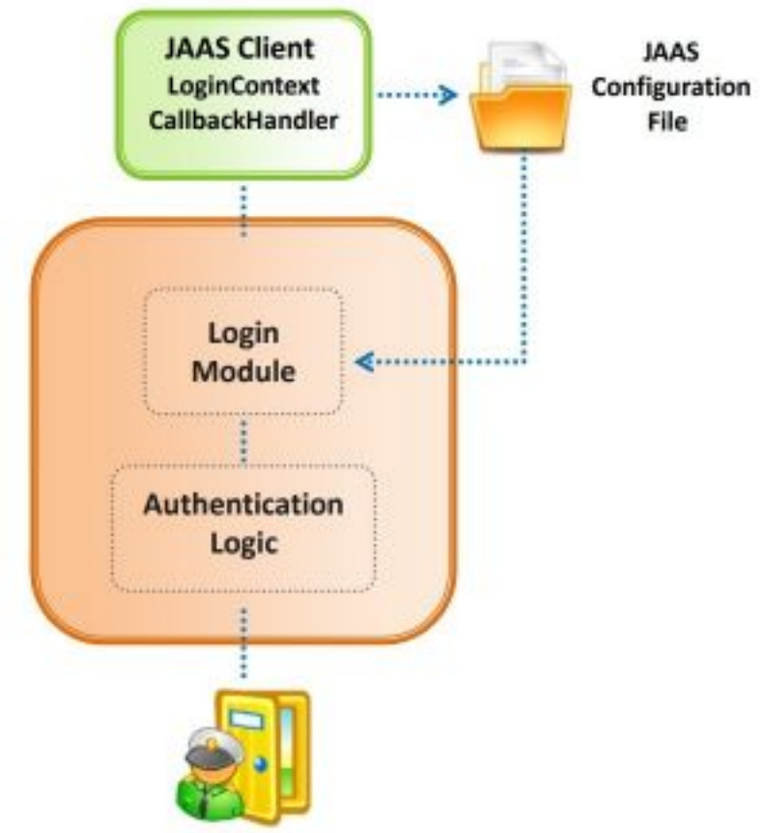
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

LoginModule Implementation in JAAS (Cont'd)

Implementation Methods

RdbmsLoginModule.java

```
1
2  import java.util.Map;
3  import javax.security.auth.Subject;
4  import javax.security.auth.callback.CallbackHandler;
5  import javax.security.auth.login.LoginException;
6
7  public class RdbmsLoginModule implements javax.security.auth.spi.LoginModule {
8
9      @Override
10     public boolean abort() throws LoginException {
11         // TODO Auto-generated method stub
12         return false;
13     }
14
15     @Override
16     public boolean commit() throws LoginException {
17         // TODO Auto-generated method stub
18         return false;
19     }
20
21     @Override
22     public void initialize(Subject arg0, CallbackHandler arg1, Map arg2, Map arg3) {
23
24     }
25     @Override
26     public boolean login() throws LoginException {
27
28         return false;
29     }
30
31     @Override
32     public boolean logout() throws LoginException {
33         // TODO Auto-generated method stub
34         return false;
35     }
36
```



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Methods Associated with LoginModule

Method	Description
`initialize()`	Called once the LoginModule is created
`login()`	Implements authentication
`commit()`	Invoked by LoginContext once it has accepted the output from all LoginModules defined in the application Principals and credentials to the Subject are assigned here
`abort()`	Executed when any LoginModule for this application fails Principals or credentials are not assigned to the Subject
`logout()`	Removes all the Principals and credentials associated with the Subject

LoginModules in J2SE

Method	Description
`JndiLoginModule`	Confirms against a directory service configured under JNDI (Java Naming and Directory Interface)
`Krb5LoginModule`	Authenticates using Kerberos protocols
`NTLoginModule`	Authenticates using the current user's NT security information
`UnixLoginModule`	Authenticates using the current user's Unix security information

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

LoginModule Example

LoginModule

SimpleAction.java

```
10
11      private String userName;
12      private String password;
13      private boolean succeeded=true;
14
15
16      public boolean login()throws LoginException, IOException
17      {
18          ECcouncilKerbCallBackHandler callbackHandler= new ECcouncilKerbCallBackHandler();
19          Callback[] CallBks = new Callback[2];
20          CallBks[0] = new NameCallback ("userName: ");
21          CallBks[1] = new PasswordCallback ("password: ",false);
22          try {
23              callbackHandler.handle(CallBks);
24              userName = ((NameCallback) CallBks[0]).getName();
25              char[] charPass = ((PasswordCallback) CallBks[1]).getPassword();
26              if (charPass == null) charPass = new char[0];
27              password = new String(charPass);
28          } catch (UnsupportedCallbackException une) {
29          }
30          if (isUserAuthentic()) succeeded = true;
31          return succeeded;
32      }
33      private boolean isUserAuthentic() {
34          // TODO Auto-generated method stub
35          return false;
36      }
```

Writable | Smart Insert | 1 : 1 | Building workspace

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Phases in Login Process

- The application calls the `login()` function of the LoginContext class that **authenticates** the Subject

- The **LoginContext** in turn calls the `login()` function of each login module in the authentication stack for application's policy configuration

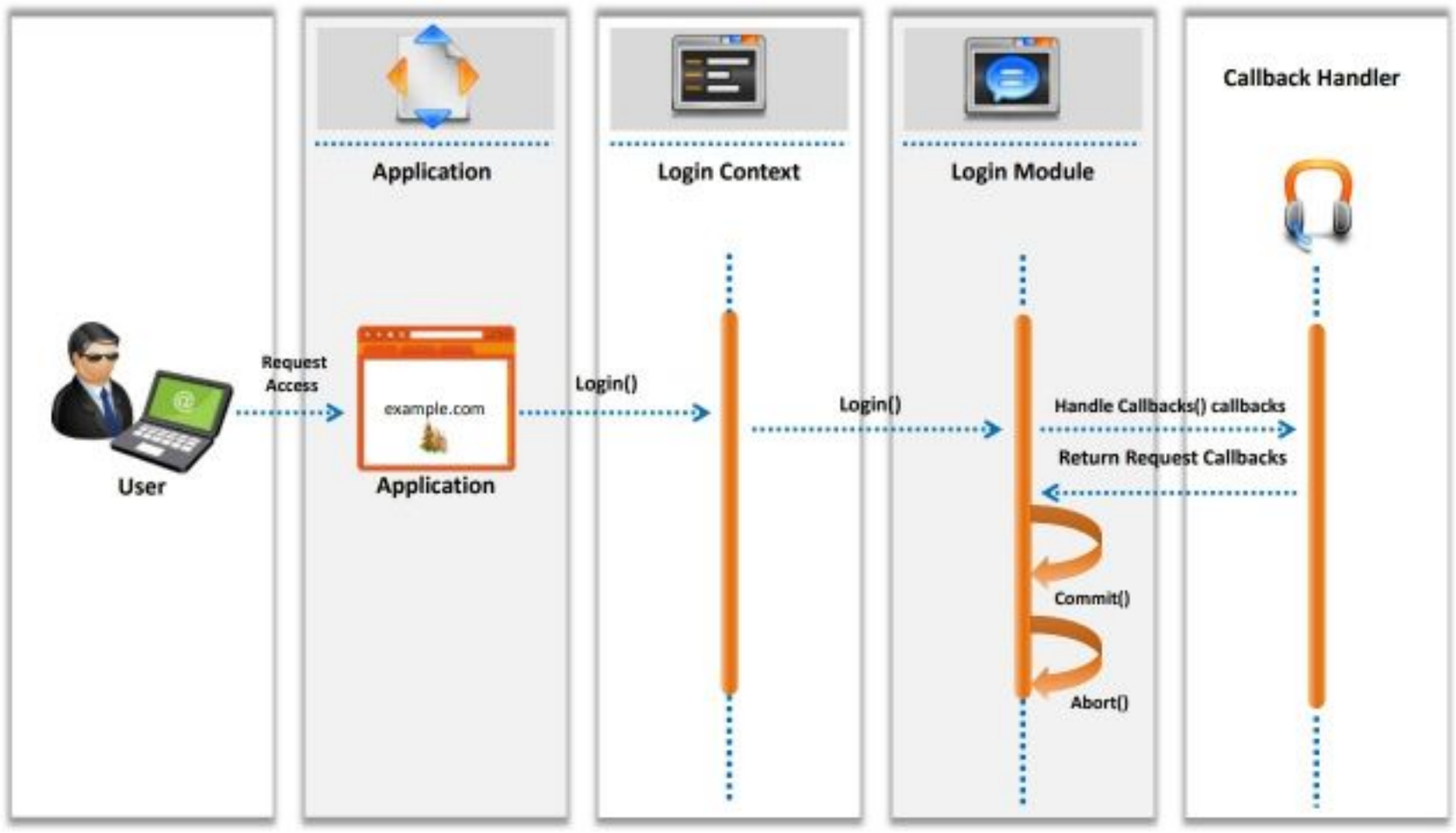
The **user authentication** is done by the login module in two phases

1. The `login()` method uses the CallbackHandler to get the authentication information from the user
2. The `commit()` method is executed if the user is successfully authenticated, and the abort() method is called if the authentication fails

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Phases in Login Process (Cont'd)



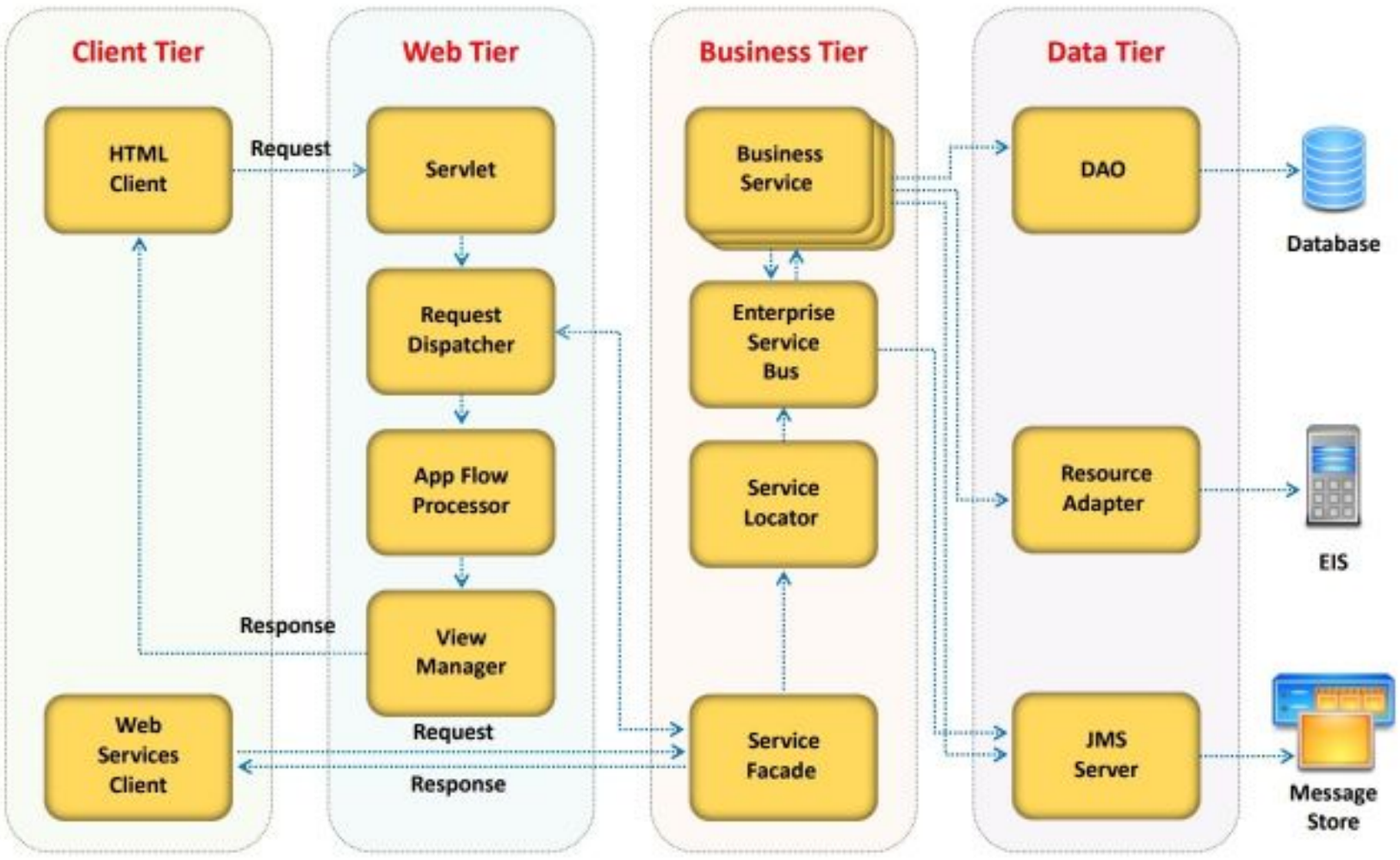
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java EE Security

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

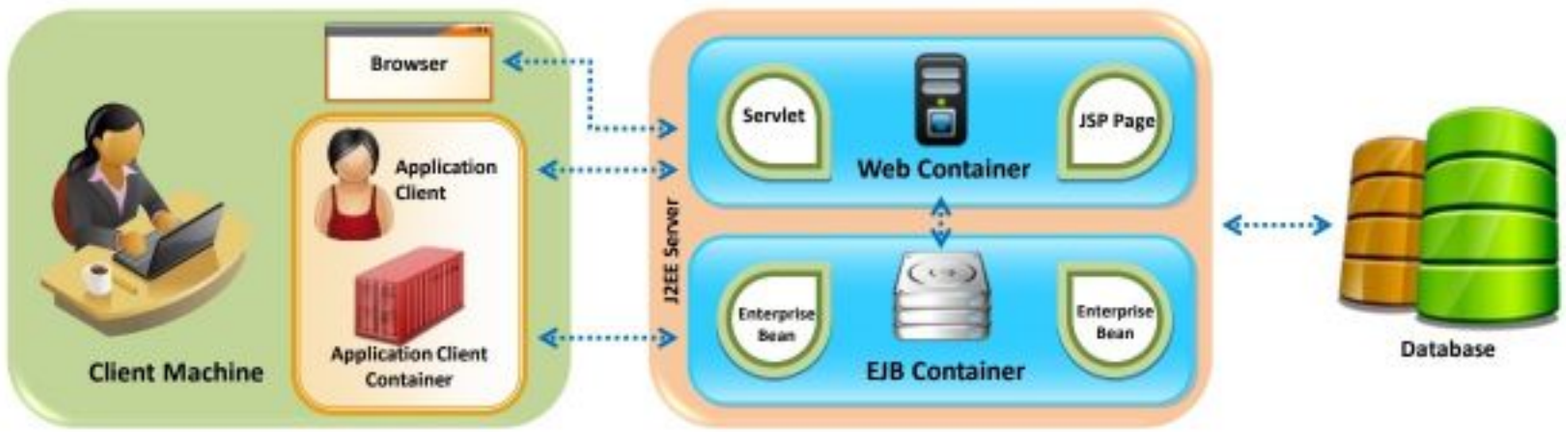
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Java EE Application Architecture



Java EE Servers as Code Hosts

- Java EE Servers are based on J2EE platform that provide support for EJB strengths, **scalability**, platform support, cross tier reuse, **code portability**
- These servers support varied types of components in a **multi-tier environment** through container
- Java EE Servers are also called **application servers** because they serve application data to clients
- Some of the popular Java EE Servers are Glassfish, **Jboss**, **Websphere**, **Weblogic**



Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Declaring Roles

Method permissions are defined using the following annotations:

@DeclareRoles

- This annotation is used to specify all the roles that are used by the application
- It also includes the roles that are not defined in the **@RolesAllowed** annotation
- The following syntax is used when declaring more than one role:
  - `@DeclareRoles({"Administrator", "Manager", "Employee"})`

@RolesAllowed("list-of-roles")

- This annotation specifies the security roles that enable access to methods in an application
- It can be defined on a class or on one or more methods

Calculate.java — DeclreRoles....

```
2
3  import org.springframework.stereotype.Controller;
4
5  @DeclreRoles("Office Admin")
6  @Controller
7  public class Calculate {
8  /*..
9      ...
10
11     ..
12     ..*/
```

Calculate.java

```
3  import org.springframework.stereotype.Controller;
4
5  @DeclreRoles({"administrator","clerk"})
6  @Controller
7  public class Calculate {
8
9  @RolesAllowed("Administrator")
10 public void setNewRate(int rate)
11 {
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Declaring Roles (Cont'd)

@DenyAll

- This annotation specifies that no security roles are allowed to run the **specified** method or methods
- These methods are eliminated from execution in the Java EE container

@PermitAll

- This annotation specifies that all **security roles** are allowed to run on the specified method or methods

Example Code for `@DenyAll` Annotation

Calculate.java

```
3  import org.springframework.stereotype.Controller;
4
5  @DeclreRoles({"administrator","clerk"})
6  @Controller
7  public class Calculate {
8
9  @RolesAllowed("Administrator")
10 public void setNewRate(int rate)
11 {
12
13 }
14 @DenyAll
15 public double convertCurrency()
16 {
```

Example: Code of `@PermitAll` Annotation

Calculate.java

```
3  import javax.annotation.security.DenyAll;
4  import javax.annotation.security.PermitAll;
5
6  import org.springframework.stereotype.Controller;
7
8  @DeclreRoles({"administrator","clerk"})
9  @Controller
10 public class Calculate {
11
12 @RolesAllowed("Administrator")
13 public void setNewRate(int rate)
14 {
15
16 }
17 @PermitAll
18 public double convertCurrency()
19 {
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

HTTP Authentication Schemes

- **The authentication schemes that are supported by HTTPClient**

Basic

- The basic authentication is defined in **RFC 2617**
- It is not secure as it passes the credentials in clear **text form**
- The security of basic authentication scheme can be enhanced by combining it with **TLS/SSL encryption**
- The `getRequestingPrompt()` method returns the Basic authentication realm as given by the server

Digest

- Digest authentication scheme is defined in RFC 2617
- It is more **secure** than the Basic authentication and suited for applications that do not require security through TLS/SSL
- The `getRequestingPrompt()` method returns the Digest authentication realm as given by the server

NTLM

- NTLM authentication scheme is defined by **Microsoft** and is more secure than Basic and less secure than Digest authentication
- NTLM is used with **proxies** or **servers** and not both at the same time

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

HTTP Authentication Schemes (Cont'd)

HTTP Negotiate (SPNEGO)

- HTTP Negotiate enables any GSS authentication to be used as a HTTP authentication protocol
- At present, it supports only **NTLM** and **Kerberos**
- Kerberos 5 Configuration: When SPNEGO calls JGSS, it in turn calls the Kerberos V5 loginmodule to perform real work

The code used to configure Kerberos using Java system property `java.security.krb5.conf.` is

```
1: java -
   Djava.security.krb5.conf=krb5.conf
   \

2: -
   Djavax.security.auth.useSubjectCre
   dsOnly=false \

3: ClassName
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Authorization Common Mistakes and Countermeasures

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Common Mistakes

1. Authorization ensures that proper **privileges** are given to authorized users

2. **Security** in authorization mainly depends on these boundaries of privileges

3. Common mistakes in authorization creep in while implementing the following security concepts like:
   - Principle of least privilege
   - Centralized authorization routines

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Common Mistakes (Cont'd)

- **Methods** to safeguard web applications against common mistakes that occur in authorization process are:
  - **Principle of least privilege**
    - Ensure **development**, **testing** and demonstration source code and environments be run at least privileges
    - Avoid web application servers running at **privileged accounts** such as administrator, root, sysman, sa, etc.
    - User accounts should have enough privileges according to their **tasks**
    - Enable web applications' access to **database** through limited accounts only
    - Ensure database access through **parameterized stored procedures**
    - Evaluate and implement **code access permissions**
  - **Protecting access to static resources on web server**
    - Do not let content save on web server in cases of **static content**
    - **Unauthorized access** to static content on the web should be prevented
    - Save sensitive files with **random names** and clean temporary files
  - **Reauthorize users or code for accessing high value activities or after idle out**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Common Mistakes (Cont'd)

- **Authorization matrix**
  - Ensure code uses built-in authorization framework or access check through **centralized authorization facilities**
- **Controlling access to protected resources**
  - Avoid using **custom authorized code**
  - Use only framework authorization and built-in platform facilities
- **Never implement client-side authorization tokens**
  - In case of a satisfactory authentication, associate session IDs with authorization **tokens**, **state** or **flags**
- **Time-based authorization should be implemented in sensitive web applications**
- **Be cautious of custom authorization controls**
  - Write **precise code**
  - In case of custom code that performs authorization, ensure fail safe authentication mechanisms and **exception handling** capability
  - Ensure authorization framework controls cover in totality to the application

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

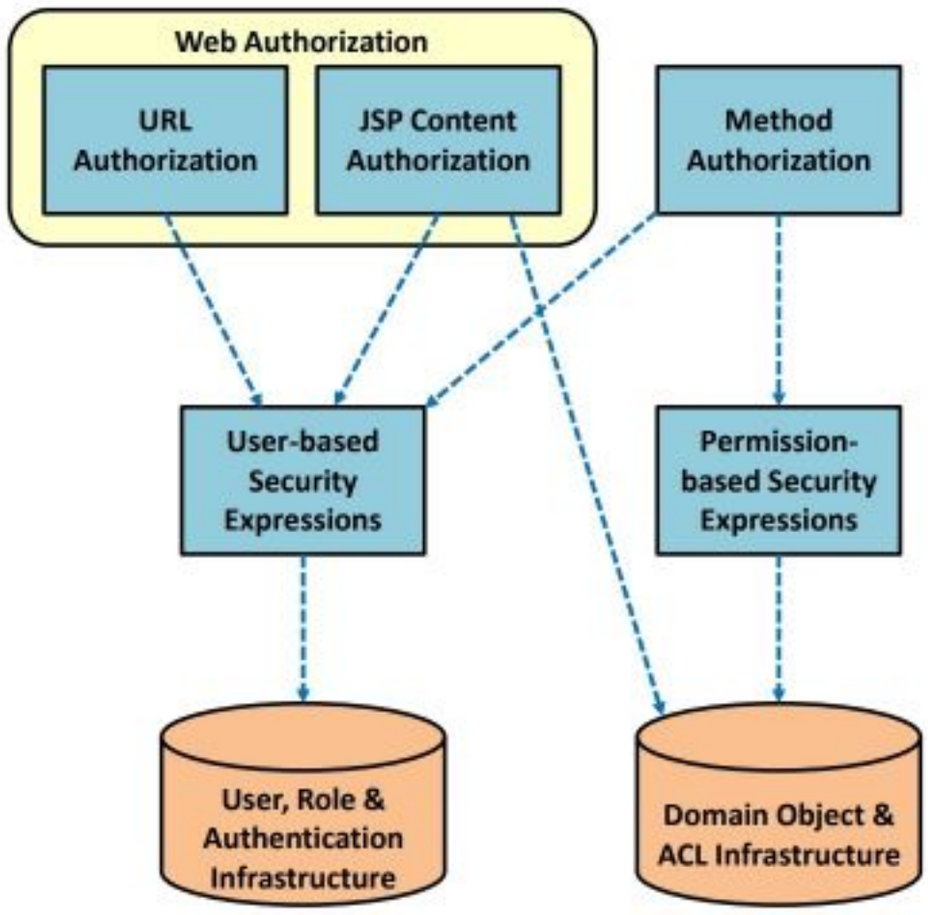
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Authentication and Authorization in Spring Security Framework

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Spring Security Framework

- Spring Security is a lightweight, customizable framework for Authentication and Authorization
- Spring Security uses **javax.servlet.Filter** to implement Authentication and Authorization
- Servlet API and Spring Web MVC can implement Spring Security Framework



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Spring Security Framework (Cont'd)

Spring Security Features

- Implement user authentication and authorization
- Provide login logout feature
- Provides role-based authorization control
- Provides link for database-based authentication and authorization
- Encrypted password support
- Supports form authentication
- Provides page-based user authentication and authorization

Authentication Types

- Http Basic Authentication
- Form-based Authentication

Authorization Types

- Web Authorization
- Method Authorization

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Spring Security Modules

Module	Description
Core(spring-security-core.jar)	Includes authentication and access-control API's
Web(spring-security-web.jar)	Includes servlet filters and web-based authentication API's . Required for web application with Web authentication and URL Authorization
Remoting(spring-security-remoting.jar)	Includes Spring Remoting API's. Required for Remote Client Applications.
Config(spring-security-config.jar)	Includes API to enable using Spring Security XML namespace as XML configuration
LDAP(spring-security-ldap.jar)	Includes API's supporting LDAP (Light weight Directory Authentication)
ACL(spring-security-acl.jar)	Includes API's for implementing domain object security using access control lists(ACL's)
CAS(spring-security-cas.jar)	Includes API's for implementing Spring Security web authentication with a CAS single sign-on server.
OpenID(spring-security-openid.jar)	Includes API's for implementing authentication against an external OpenID server

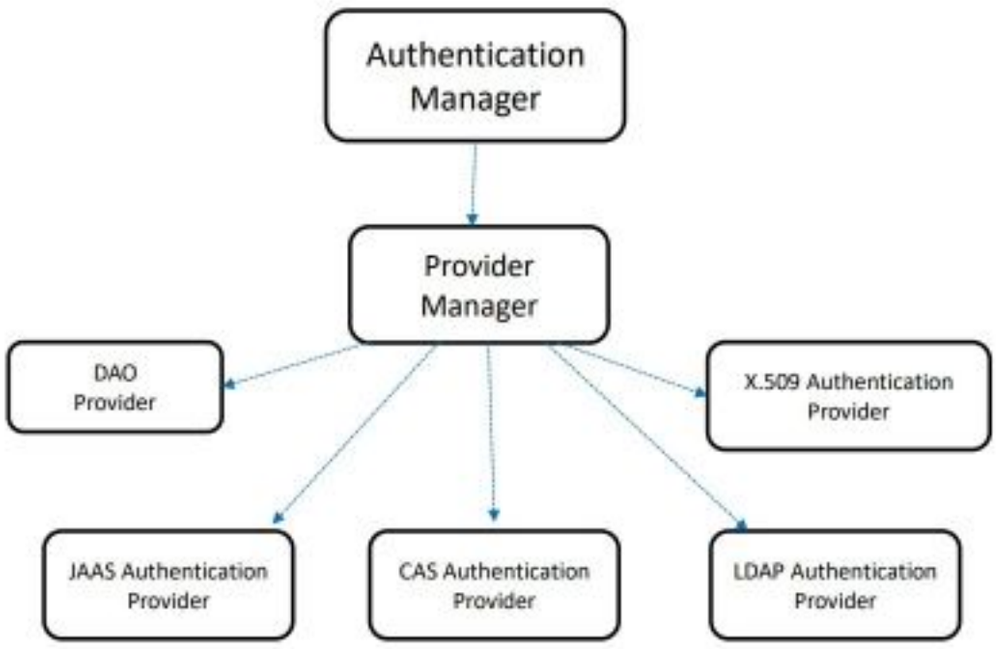
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Spring Authentication

Authentication Process

- Spring Security Supports a variety of pluggable Authentication Providers, e.g. , LDAP, X.509 (Certificates), Database (JDBC), JAAS, OAuth
- **AuthenticationManager** (Interface for processes all authentication requests) is the core for processing all authentication requests
- **AuthenticationProvider** (Interface for Authentication process) is used to define authentication provider
- AuthenticationProvider implements **UserDetailsService** (Service interface for fetching UserDetails object), which is responsible to get the User details
- The retrieved user details are **matched** with the provided username/ password while login



Authentication Manager and Provider Manager

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Storing Username and Password

- **‹authentication-provider> tags** links to **DaoAuthenticationProvider** and calls the **UserDetailsService** implementation

Storing User Details in Application Context

- When the number of users for the application are less, we can directly add the login and passwords in XML
- **org.springframework.security.core.userdetails.memory.InMemoryDaoImpl** bean is initialized for ‹user-service›

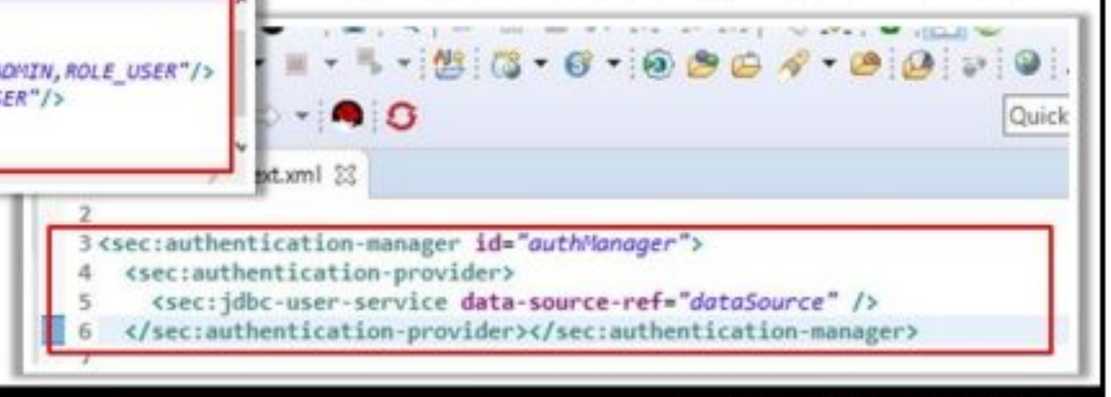
Storing User Details in Database

- When the number of users for the application are more, the login and passwords are stored in the database
- **org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl** class is used

Configuration to Store Login Details in Application Context



Configuration to Store Login Details in Database



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Storing Username and Password (Cont'd)

Storing User Details in Custom User Table

- To store and retrieve Login details from a custom table, configure the fetch queries, to be executed for getting the user credentials

Configuration to alter fetch query to get credentials from USER table with id, username, password columns and Role table which has username, role columns

ApplicationContext.xml

```
3 <sec:authentication-manager id="authenticationManager">
4 <sec:authentication-provider>
5 <!-- TBD <password-encoder hash="md5"/> --><sec:jdbc-user-service id="userDetailsService"
6 data-source-ref="dataSource"
7 users-by-username-query="SELECT username, password, true as enabled FROM USER WHERE username=?"
8 authorities-by-username-query= "SELECT user.username, role.role as authorities FROM ROLE role, USER
9 WHERE role.user_id=user.id and user.username=?"/>
10 </sec:authentication-provider>
11 </sec:authentication-manager>
```

Design Source

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Securing Authentication Provider

Encode Password using -SHA1

- While storing user details in **Application Context**, the password is configured as plain text
- Encode the password using **SHA1** using **<password-encoder>** element

Vulnerable Authentication Provider Configuration

ApplicationContext.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <sec:authentication-manager id="authManager">
4 <sec:authentication-provider>
5 <sec:user-service>
6 <sec:user name="admin" password="password" authorities="ROLE_ADMIN,ROLE_USER"/>
7 <sec:user name="user" password="password" authorities="ROLE_USER"/>
8 </sec:user-service></sec:authentication-provider>
9 </sec:authentication-manager>
```

Secure Authentication Provider Configuration

ApplicationContext.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <sec:authentication-manager id="authManager">
4 <sec:authentication-provider>
5 <password-encoder hash="sha" />
6 <sec:user-service>
7 <sec:user name="admin" password="password" authorities="ROLE_ADMIN,ROLE_USER"/>
8 <sec:user name="user" password="password" authorities="ROLE_USER"/>
9 </sec:user-service></sec:authentication-provider>
10 </sec:authentication-manager>
11
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Implementing HTTP Basic Authentication

- Defining **‹http-basic›** defines a **BasicAuthenticationFilter** filter
- On successfully authentication of the user, the Authentication object is added to Spring **SecurityContext**
- **SecurityContextHolder** class is used to access the security context

Configuring HTTP Basic Authentication

spring-bean.xml | root-contex... | spring-secu...

```
<http auto-config="true">
    <intercept-url pattern="/admin*" access="ROLE_ADMIN" />
    <logout logout-success-url="/admin" />
</http>
```

BasicAuthenticationFilter Bean Declaration

spring-bean.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

 <bean id="basicAuthenticationFilter"
 class="org.springframework.security.web.authentication.www.Basic AuthenticationFilter">
 <property name="authenticationManager" ref="authenticationManager"/>
 <property name="authenticationEntryPoint" ref="authenticationEntryPoint"/>
</bean>
<bean id="authenticationEntryPoint"
 class="org.springframework.security.web.authentication.www.BasicAuthenticationEntryPoint">
 <property name="realmName" value="Name Of Your Realm"/>
</bean>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Form-based Authentication

- **Spring Security includes a login form for authentication by default**

Configuring Form-based Authentication

- **global-method-security:** Will enable the **@PreAuthorize**, **@PostAuthorize** annotations tag
- **session-management:** Includes **SessionManagementFilter** to Spring Security's chain of filters
- **form-login:** Configure **Form-based Authentication**
  - **default-target-url:** Redirects to specified page on successful authentication
  - **authentication-failure-url:** Redirects to specified page when authentication fails
  - **always-use-default-target:** When set to true will redirect user to the specified page after login
  - **authentication-success-handler-ref:** Gets executed on successful authentication
  - **authentication-failure-handler-ref:** Gets executed on authentication fails
- **username-parameter:** Link the defined username property of **login.jsp** page
- **password-parameter:** Link the defined password property of **login.jsp** page

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Form-based Authentication (Cont'd)

Sample Login Form Configuration

spring-bean.xml

```
8 <global-method-security pre-post-annotations="enabled" />
9 <http use-expressions="true" access-denied-page="/403" disable-url-rewriting="true">
10 <session-management invalid-session-url="/login?time=1" ><concurrency-control max-sessions="1" expired-url="/login?time=1"/>
11 </session-management>
12 <form-login login-page="/login" login-processing-url="/login.do" default-target-url="/" always-use-default-target="true"
13 authentication-failure-url="/login?err=1"/>
14 </http>
15 </beans:beans>
```

Design Source

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Implementing Digest Authentication

- Implementing Digest Authentication will encrypt the user password using hashing algorithms before sending it to the server
- MD5, SHA, BCrypt, SCrypt and PBKDF2WithHmacSHA1 are the hashing algorithms used for hashing password
- Digest contains **"nonce"**
  - Prevents plaintext attacks for retrieving plain text from cryptographic hash functions
  - It contains timestamps and prevents replay attacks
- **Example :** "nonce" value format generated by server
  - **expirationTime**: Expiration time in millisecond
  - **key**: Its private key to protect "nonce" changes

DigestAuthenticationFilter Configuration

spring-bean.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:context="http://www.springframework.org/schema/context"
5     xsi:schemaLocation="
6     http://www.springframework.org/schema/beans
7     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
8     http://www.springframework.org/schema/context
9     http://www.springframework.org/schema/context/spring-context-3.0.xsd">
10
11 <bean id="digestFilter" class=
12  "org.springframework.security.web.authentication.www.DigestAuthenticationFilter">
13  <property name="userDetailsService" ref="jdbcDaoImpl"/>
14  <property name="authenticationEntryPoint" ref="digestEntryPoint"/>
15  <property name="userCache" ref="userCache"/>
16 </bean>
17 <bean id="digestEntryPoint" class="org.springframework.security.web.authentication.www.DigestAut
18  <property name="realmName" value="Contacts Realm via Digest Authentication"/>
19  <property name="key" value="acegi"/>
20  <property name="nonceValiditySeconds" value="10"/>
21 </bean>
22 </beans>
```

Example : Unauthorized Response

```
HTTP/1.1 401 Unauthorized Server: Apache-Coyote/1.1 Set-Cookie:
JSESSIONID=CF0233C...; Path=/spring-security-mvc-digest-auth/;
HttpOnly WWW-Authenticate: Digest realm="Contacts Realm via
Digest Authentication", qop="auth",
nonce="MTM3MzYzODE2NTg3OTo3MmYxN2JkOWYxZTc4MzdmMzBiN2Q0YmY0ZTU0
N2RkZg==" Content-Type: text/html;charset=utf-8 Content-Length:
1061 Date: Fri, 12 Jul 2013 14:04:25 GMT
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Security Expressions

- Spring security includes **Spring Expression Language** (SpEL) security expressions for authorization
- Security expressions are evaluated against a context-dependent "root object" called **SecurityExpressionRoot**
- In web context, the root object is called **WebSecurityExpressionRoot**, which is derived from SecurityExpressionRoot

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Security Expressions (Cont'd)

User Security Expression Terms and Predicates

Term	Description
Authentication	Current user's Authentication object, derived from SecurityContext
Principal	Current user's principal object, derived from Authentication object

Expression	Description
hasRole([role])	User has the specified role then return "true"
permitAll	Always true
denyAll	Always false
isAnonymous()	Return true if user is anonymous
isRememberMe()	Return true if user is authenticated using Remember-Me option
isAuthenticated()	Return true if the user is not anonymous user
isFullyAuthenticated()	Return true if the user is not anonymous user and is not authenticated using Remember-Me option

WebSecurityExpressionRoot Terms

Term	Description
request	Denotes the HttpServletRequest
hasIpAddress(ipAddr)	Returns true if the client IP address same as the specified IP address

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

URL-based Authorization

- To implement URL-based Authorization, use **‹intercept-url›**
- <intercept-url> contains the following attributes
  - **Pattern**: Specifies the url patterns
  - **Access**: Contains the list of user roles that can access the url
  - **Method**: Optional parameter specifying a HTTP method for authorization
- The specified urls to be intercepted are sent as **metadata** to **FilterSecurityInterceptor**
- Ensure that the specified url-patterns in the <intercept-url> **ends with "*"**, otherwise an attacker can **pass parameters** to the Url to bypass the Authorization rule

Enable expression-based web URL authorization

Sample intercept-url configuration in XML

```
spring-security.xml
4  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
5  http://www.springframework.org/schema/security
6  http://www.springframework.org/schema/security/spring-security-3.0.3.xsd"
7
8  <http auto-config="true"
9   use-expressions="true"
10  access-decision-manager-ref="accessDecisionManager">
11     <intercept-url pattern="/admin*" access="ROLE_ADMIN" />
12     <intercept-url pattern="/app/*" access="ROLE_USER" />
13     <form-login login-page="/Mylogin.jsp"
14     default-target-url="/app/messagePost"
15      authentication-failure-url="/Mylogin.jsp?error=true"/>
16     <logout logout-success-url="/admin" />
17 </http>
Source  Namespaces  Overview  beans  sec
```

Sample intercept-url Configuration without XML

```
*spring-security.xml
20 <beans:bean id="filterSecurityInterceptor"
21 class= "org.springframework.security.web.access.intercept.FilterSecurityInterceptor">
22 <beans:property name="authenticationManager" ref="authenticationManager"/>
23 <beans:property name= "accessDecisionManager" ref="accessDecisionManager"/>
24 <beans:property name="securityMetadataSource">
25 </beans:property>
26 <sec:intercept-url pattern="/app/messageList*" access="ROLE_USER,ROLE_ANONYMOUS"/>
27 <sec:intercept-url pattern="/app/messagePost*" access="ROLE_USER"/><sec:intercept-url
28 pattern="/app/messageDelete*" access="ROLE_ADMIN"/>
29 <sec:intercept-url pattern="/app/*" access="ROLE_USER"/>
30 </sec:filter-security-metadata-source></property>
31 </beans:bean>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

JSP Page Content Authorization

- Authorizing **JSP page** content based on the logged in users status, role, etc.
- Include the following to implement web URL authorization

```
home.jsp
1 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2 <%@ taglib prefix="security"  uri="http://www.springframework.org/security/tags" %>
```

Include Tag Library to Implement Web URL Authorization

Tag Library Attributes List

Tag attributes	Description
access	Show tag content when specified Web security access expression is true
url	Show tag content when the user has access permissions for the specified url
Method	Limit authorization to the specified HTTP method (e.g., GET, POST, PUT, DELETE)

Tag Library Tags List

Term	Description
<security:authentication>	Fetch the current Authentication object to the JSP
<security:authorize>	Display or hide the tag content based on the current user matches the specified condition
<security:accesscontrollist>	Display or hides the tag content based on current user permission matches the specified domain object

Display Login Link to Unauthenticated Users

```
home.jsp
10 <body>
11 <h1>
12 <security:authorize access="isAnonymous()">
13   <a href="${loginUrl}">Log in</a>
14 </security:authorize>
15
```

Display New Course Link to Users with Role "Instructor"

```
home.jsp
10 <body>
11 <h1>
12 <security:authorize access="hasRole('instructor')">
13   <a href="${CreateUser}">New Course</a>
14 </security:authorize>
15
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

JSP Page Content Authorization with Domain Object's ACL

- Authorizing JSP page content based on the logged in users' permission on domain objects
- <security:accesscontrollist> tag displays or hides content of JSP page based on permissions of logged in user for a domain object's ACL
- Expressions are not used by <security:accesscontrollist> tag

Tag attributes	Description
hasPermission	Show or hide content when at least one of the specified permissions exist (list of numeric permissions)
domainObject	Domain object for checking the specified permission

Display Edit link on blog when the logged in user is Admin or had edit permission on the blog object

*home.jsp

```
10 <body>
11 <h1>
12 <security:accesscontrollist
13    hasPermission="2,16" domainObject="${blog}">
14    <a href="${editUrl}">Edit message</a>
15 </security:accesscontrollist>
```

Design | Preview

Method Authorization

- Spring security use **security expression annotations** to protect methods and classes
- Include the global-method-security namespace to implement @Pre/@Post annotations

XML Configuration for @Pre/@Post Annotations

spring-security.xml

```
8 <global-method-security pre-post-annotations="enabled">
9 </global-method-security>
```

JAVA Configuration for @Pre/@Post Annotations

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class JSecurityConfig extends
WebSecurityConfigurerAdapter {   ...}
```

@PreAuthorize

- **@PreAuthorize** ensures that the method is invoked only when the specified expression is true

@PostAuthorize

- **@PostAuthorize** ensures that the method returns value only when the specified expression is true

Only Guest User can Create guestlist

spring-security.xml | User.java | HomeController.java | guestlist.java

```
        return "home";
    }

    @PreAuthorize("hasRole('ROLE_guest')") public void create(guestlist guest) {
    }
```

Only Admin can read the Message

HomeController.java

```
@PostAuthorize("hasRole('admin') or  hasPermission(filterObject, admin)")
        public Message getMessage(long id) {
            return msg;  }
```

- **Implementing Filter using @PreFilter and @PostFilter**
  - Filter collections and arrays based on the specified expression

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Configuring Anonymous Login

- Spring Security creates by default an anonymous role, which can be used for providing anonymous login
- Using '**ROLE_ANONYMOUS**' or '**IS_AUTHENTICATED_ANONYMOUSLY**' gives anonymous users access to any page
- The default roles for anonymous users can be altered by overriding the default configuration

Configuring Custom Anonymous Role

spring-security.xml

```
29  <sec:intercept-url pattern="/login.jsp*" filters="none"/>
30  <sec:intercept-url pattern="/*" access="ROLE_USER"/>
31  <!-- Defines a custom role in place of ROLE_ANONYMOUS.
32  ROLE_ANONYMOUS will no more work, use ROLE_GUEST instead of it-->
33  <sec:anonymous username="guest" granted-authority="ROLE_GUEST" />
34 </sec:http>
35 <p style="text-align: justify;">Here is the how the underlying filter can be defined
36 <sec:custom-filter position="ANONYMOUS_FILTER" ref="anonymousFilter" />
37 <bean id="anonymousFilter" class="org.springframework.security.web.authentication.Ano
38     <property name="userAttribute" value="ROLE_GUEST" />
39 </bean>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Logout Feature Configuration

- Spring security provides **handler** to support logout feature
- **/j_spring_security_logout** is the default log-out url
- Define **logout-url** *attribute to* customize default logout url
- Define **logout-success-url** *to redirect to a page after logout*
- For conditional redirection to different pages after logout, implement **LogoutSuccessHandler** and link it to **‹logout›** element

Configuring Logout Feature

spring-security.xml

```
13    <http auto-config="true">
14        <logout logout-success-url="/app/messageList" />
15    </http>
16
```

Source | Namespaces | Overview | beans | sec

Implementing LogoutSuccessHandler

spring-security.xml

```
13    <http auto-config="true">
14        <logout logout-success-url="/app/messageList"
15        success-handler-ref="customLogoutSuccessHandler"/>
16    </http>
17
```

Source | Namespaces | Overview | beans | sec

Define Logout Filter

spring-security.xml

```
32 <bean id="logoutFilter" class="org.springframework.security.web.authentication.logout.LogoutFilter">
33 <constructor-arg value="/pages/Security/logout.html" /> <constructor-arg> <list>
34 <bean class="org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler"/>
35 </list> </constructor-arg> <property name="filterProcessesUrl" value="/j_logMeOut"/> </bean>
36
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Remember-Me Authentication

- Spring security supports **Remember-Me feature**
- Once the Remember-Me feature is enabled, a **cookie base64(username + ":" + expirationTime + ":" + md5Hex(username + ":" + expirationTime + ":" password + ":" + key))** is sent to the browser after successful authentication at login
- Browser sends the received authentication **cookie** with every request to the server
- Spring security retrieves the **password** from the database for the logged in username
- Evaluates **md5Hex()** for the username, password along with expiration time and key and matches it with the supplied cookie
- If the evaluated values match with the supplied cookie, the user automatically gets logged in

Enabling Remember-Me Authentication

spring-security.xml

```
13  <http auto-config="true">
14      <logout logout-success-url="/app/messageList"
15      success-handler-ref="customLogoutSuccessHandler"/>
16
17      <remember-me key="APPLICATIONKEY"/>
18  </http>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Integrating Spring Security with JAAS

- Spring Security contains a **package** to delegate authentication requests to JAAS
- Spring Security's authentication process takes the **username** and **password** provided by the user and stores it in the authentication object
- Each authentication object contains one **principle**

JAAS package components

- Authentication
- AuthenticationProvider
- LoginContext
- AuthorityGranter
- JaasAuthenticationToken

JAAS based Classes and Interfaces

- org.springframework.security.authentication.jaas
- AbstractJaasAuthenticationProvider
- AuthorityGranter
- DefaultJaasAuthenticationProvider
- DefaultLoginExceptionResolver
- JaasAuthenticationCallbackHandler
- JaasAuthenticationToken
- JaasGrantedAuthority
- JaasNameCallbackHandler
- LoginExceptionResolver
- SecurityContextLoginModule

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Spring JAAS Implementation

Implement AuthorityGranter Interface

RoleGranter.java

```
1  package com.myproject.webapp;
2
3  import java.security.Principal;
4  import java.util.Collections;
5  import java.util.Set;
6
7  import org.springframework.security.authentication.jaas.Auth
8
9  public class RoleGranter implements AuthorityGranter{
10
11     @Override
12     public Set<String> grant(Principal principal){
13         if (principal.getName().equals("admin"))
14         return Collections.singleton("ADMIN");
15         else
16         return Collections.singleton("ENDUSER");
17     }
18 }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Spring JAAS Implementation (Cont'd)

Implementation of JaasGrantedAuthority and UserPrinciple

ItsSecureController.java

```
1  package jaaSimpl;
2
3  import java.nio.file.attribute.UserPrincipal;
4
5  import org.springframework.security.authentication.jaas.JaasGrantedAuthority;
6  import org.springframework.security.core.Authentication;
7  import org.springframework.security.core.context.SecurityContextHolder;
8  import org.springframework.ui.ModelMap;
9  import org.springframework.web.bind.annotation.RequestMapping;
10
11 public class ItsSecureController {
12
13    @RequestMapping(value="HomeUser/Home")
14    public String getUser(ModelMap model)
15
16    Authentication auth= SecurityContextHolder.getContext().getAuthentication();
17
18    JaasGrantedAuthority jaasGrantedAutherity=(JaasGrantedAuthority)(auth.getAuthorities());
19    UserPrincipal usrPrincipal =(UserPrincipal)jaasGrantedAutherity.getPrincipal();
20        model.addAttribute("userPrincipal",usrPrincipal);
21    return "HomeUser/Home";
22    }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Defensive Coding Practices against Broken Authentication and Authorization

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Do Not Store Password in Java String Object

- Password stored in a Java String object exists in the memory until the process ends or garbage collector clears the memory
- Password will remain in the free memory heap as long as the memory space is not reused. Hence, password stored in String objects are vulnerable to snooping as long as they exist in the memory
- Password in memory can be moved to disk's swap space when system memory is low. This makes the password vulnerable to disk block snooping
- To protect passwords from such vulnerabilities, the passwords should be stored in char arrays and then **cleared** after use

Vulnerable Code

User.java

```
1
2   String pwd= request.getParameter("pwd");
3
```

Secure Code

User.java

```
3
4   char[] mypass=request.getParameter("pwd").toCharArray();
5
```

- Storing password in Java String Object
- Storing password in Char array

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Avoid Cookie Based Remember-Me Use Persistent Remember-Me

- Implementing **TokenBasedRememberMeServices** stores the MD5 hash of user password in the cookie
- Cookie-based Remember-Me functionality is vulnerable to cookie attacks
- Implementing **PersistentTokenBasedRememberMeServices** stores a unique persistent token in the database
- This token is regenerated and stored every time when a user logs in using persisted Remember-Me functions
- Hence, **PersistentTokenBasedRememberMeServices** approach prevents brute force attack

Vulnerable Code

spring-security.xml

```
<http auto-config="true">
    <logout logout-success-url="/app/messageList"
    success-handler-ref="customLogoutSuccessHandler"/>
    <remember-me key="APPLICATIONKEY"/>
</http>
```

- Implementing **TokenBasedRememberMeServices** for Remember-Me functionality

Secure Code

spring-security.xml

```
5  <http auto-config="true">
6      <logout logout-success-url="/app/messageList"
7      success-handler-ref="customLogoutSuccessHandler"/>
8      <remember-me key="APPLICATIONKEY" data-source-ref="dataSource"
9       token-validity-seconds="86400"/>
0  </http>
```

- Implementing **PersistentTokenBasedRememberMeServices** for Remember-Me functionality

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Implement Appropriate Session Timeout

- Ensure that the **timeout** of session is not too long
- Session is properly **invalidate** or **terminated**
- Set appropriate session time out
- Use **Tomcat/config/web.xml** to set session time out at server level
- Use **$Tomcat/webapps/myproje/WEB-INF/web.xml** to set session time out at application level

Vulnerable Code

web.xml

```
46    </web-resource-collection>
47    </security-constraint>
48    <session-config>
49    <session-timeout>-1</session-timeout>
50    </session-config>
51
```

Tree Source

- Session time out is set to unlimited duration

Secure Code

web.xml

```
46    </web-resource-collection>
47    </security-constraint>
48    <session-config>
49    <session-timeout>15</session-timeout>
50    </session-config>
51
```

Tree Source

- Session time out is set to 15 mins

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Implement Appropriate Session Timeout (Cont'd)

Secure Code : Setting Different Session Timeout for Different Users

web.xml | LoginServlet.java

```
6    {
7        HttpSession session=req.getSession(true);
8
9        if(req.isUserInRole("ADMIN"))
0        {
1            session.setMaxInactiveInterval(30*60);
2        }else
3        {
4            session.setMaxInactiveInterval(15*60);
```

Secure Code : Setting Session Timeout Programmatically

ppHttpSessionListener.java

```
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

public class AppHttpSessionListener implements HttpSessionListe

    @Override
    public void sessionCreated(HttpSessionEvent event) {

        event.getSession().setMaxInactiveInterval(15 * 60);

    }

    @Override
    public void sessionDestroyed(HttpSessionEvent event) {
            //destroy session
```

Note: `setMaxInactiveInterval` will set session time out separately for Admin and user roles

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Prevent Session Stealing by Securing SessionID Cookie

- Session stealing happens when data is transferred over network as **clear-text.** It can be prevent by setting authentication on server
- Set AuthCookieEnabled="**true**" in the WebServer element in config.xml of WebLogic Server
- When **AuthCookieEnabled** is set to **true**, while authenticating using HTTPS, the server sends a new secure cookie, _WL_AUTHCOOKIE_JSESSIONID
- When session id uses default JSESSIONID, it can be stolen because it is not in encrypted format, but _WL_AUTHCOOKIE_JSESSIONID is in encrypted format over HTTPS

Setting AuthCookieEnabled in config.xml

```
<WebServer Name="myserver" AuthCookieEnabled="true"/>
```

Note: By default, AuthCookieEnabled is set to true

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Development Checklists: Broken Authentication and Session Management

Check whether your **server** stores or validates **user passwords**
Do not transmit passwords over the **network** in **clear text** format
Check whether the security mechanism has provisions to **create**, **modify**, **delete**, and **validate** user passwords
Ensure that the client provides **authentication** information only after the **server** has provided its own authentication **credential** for **verification**
Check if the application ever reissues a password given to it by a **client** to a **third-party** application
Check if the application supports **Kerberos** and other **authentication** methods
Avoid allowing the application to support **unauthenticated** (guest) access
Verify if the application uses **Open Directory** for all **authentications**
Implement **built-In session management**
Ensure **login** page is **SSL-protected**
To prevent session fixation, **Invalidate** the **HttpSession** before login. Take proper care to invalidate session, and clear all authentication related data on **Logout**
Set appropriate **session timeout** to ensure that sessions are not reused by attackers
Ensure proper implementation for **forgot password, reset password and Remember-Me** functionality as they can compromise security
Implement **multiple authentication** systems (like token authentication, OTP) along with Username password checking
Check the user **access privileges** before accessing resources
Validate the URL before accepting or submitting

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Module Summary

- The main purpose of authentication is to verify a user's access to a protected segment of the web application
- Authentication is of two types: basic authentication and form-based authentication
- J2EE container services provide application tiers, and components with authentication and authorization facilities identifying service providers and callers
- Authentication in a web application is role-based, i.e., a user needs to be assigned a role in order to access the web application, e.g., customer, developer or a manager
- Authorization is the process that controls access rights of principals to system resources that include: Access to users, Access to processes, Access to machines
- Authorization shows "who is executing" "where is the code is residing" "who is the owner of the code"
- Access control comprises three models: discretionary access control (DAC), mandatory access control (MAC) and role-based access control (RBAC)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

JAVA

C | ASE

Certified | Application Security Engineer



Module 06

Secure Coding Practices for Cryptography

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

Module Objectives

1	Understand Fundamental Concepts and Need of Cryptography in Java
2	Discuss Encryption and Secret Keys
3	Discuss Implementation of Cipher Class
4	Discuss Implementation of Digital Signatures
5	Discuss Implementation of Secure Socket Layer (SSL)
6	Discuss Secure Key Management
7	Discuss Implementation of Digital Certificates
8	Discuss Implementation of Hashing
9	Explain Java Card Cryptography
10	Explain Crypto Module in Spring Security
11	Learn Dos and Don'ts in Java Cryptography

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java Cryptography

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Need for Java Cryptography

- When a secret needs to be kept, you have to find a way to protect it from:
  - Misuse of data
  - Unauthorized access
  - Data theft
  - Deleting of data
- Protecting applications against attacks and maintaining privacy is a perplexing task for organizations due to **decentralized operations**, **diverse Internet users** and **distributed systems**
- Data collected through attacks against applications can be used for creating unauthorized profiles, deleting authorized profiles, misuse and manipulation of sensitive data, jeopardizing trade secrets and business transactions
- Due to threats to identity and privacy, IT developers are now focusing on providing security solutions that **enhance**, **support**, and **manage** associated risks with **data storage**, **access**, and **usage**
- Cryptography uses **ciphers** to encrypt data, **digital certificates** to authenticate trusted sites, and **digital signatures** to ensure that data is not compromised or tampered

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java Security with Cryptography

- Cryptography known as **secret writing**, can convert data into a scrambled code that is **encrypted**, **decrypted** and sent across a private or public network
- Cryptography deals with **security** issues in accordance with:
  - Privacy
  - Integrity
  - Authentication
  - Non-repudiation
- The Java platform, through APIs, addresses major security areas such as **cryptography**, authentication, **public key infrastructure**, secure communication and access control
- With these APIs, developers can easily design and write application code by integrating both **low-level** and **high-level security functions**
- Java platform offers cryptographic operations using **APIs** such as
  - JCA (Java Cryptography Architecture)
  - JCE (Java Cryptography Extension)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Java Security with Cryptography (Cont'd)

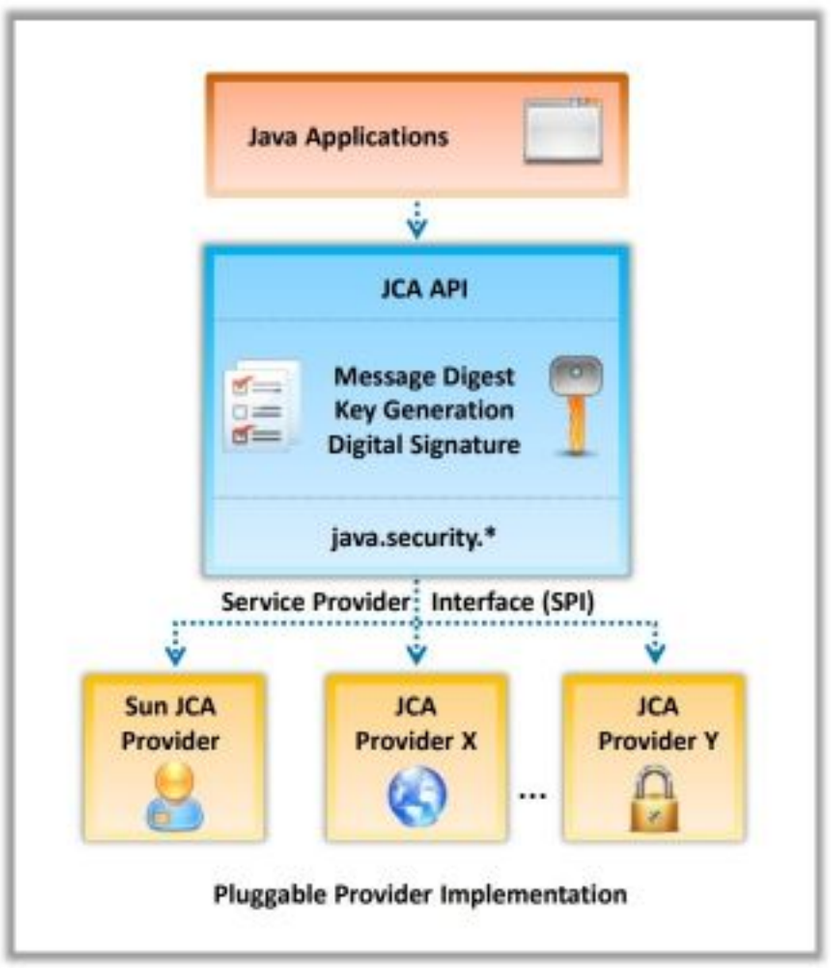
- **Design principles for JCA and JCE**

1. Algorithm Independence
2. Algorithm Extensibility
3. Implementation Independence
4. Implementation Interoperability

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java Cryptography Architecture (JCA)

- JCA provides an **architectural framework** for executing the main cryptographic services in the Java platform
- Java cryptographic operations are realized through **engines classes** that abstract with a particular cryptographic concept
- JCA classes are located in **java.security package**
- JCA supports many of the standard **algorithms** such as RC4, RC2, PKCS#5, SHA, Triple DES, AES, DSA, RSA, etc.



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Java Cryptography Architecture (JCA) (Cont'd)

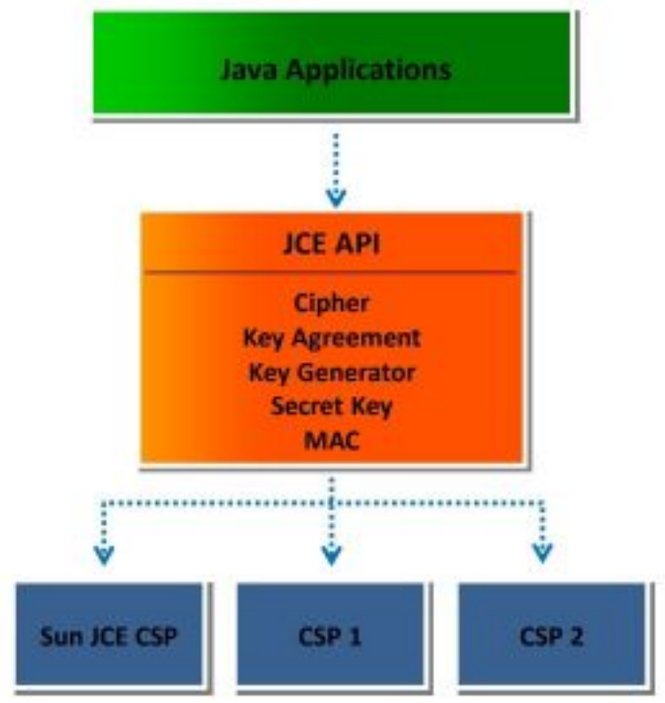
Java Cryptography Architecture Engine Includes

- Message Digests (create hash value)
- Digital Signatures (create signatures)
- Key Pair Generator (creates a pair of keys)
- Key Factories (break down a key)
- Key Store (manages and stores a key)
- AlgorithmParameters (encoding and decoding)
- AlgorithmParametersGenerator (generating parameters)
- CertificateFactory (generates public key certificates)
- CertPathBuilder (creates relationship chains between certs)
- CertStore (manages and stores certs)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java Cryptography Extension (JCE)

- JCE includes classes for encryption and decryption, used for **ciphering** and **deciphering** functionality
- JCE engines include
  - Ciphers (manage encryption/decryption)
  - KeyGenerator (creates secret keys used by ciphers)
  - SecretKeyfactory (operates on SecretKey objects)
  - Key agreement (key agreement protocol)
  - MAC (message authentication code functionality)
- JCE engines are located in **javax.crypto package**
- SunJCE provider supports **algorithms** such as DES, DESede, Triple-DES, Blowfish, RSA, PBEWithMD5AndDES, HmacMD5, HmacSHA1, etc.



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Encryption and Secret Keys

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Attack Scenario: Inadequate/Weak Encryption

1. The data transmitted across the network can be intentionally or unintentionally **modified** or **manipulated** hence it cannot be considered as secure
2. Inadequate or weak encryption can lead to data breaches where an **intruder** can exploit prevailing **vulnerabilities** between server and client communications
3. In weak encryption based attacks, an attacker can **decrypt sensitive data** using brute force attacks
4. Weak encryption keys can allow attackers to perform reverse engineering, man-in-the-middle attacks, spoof content, phishing attacks

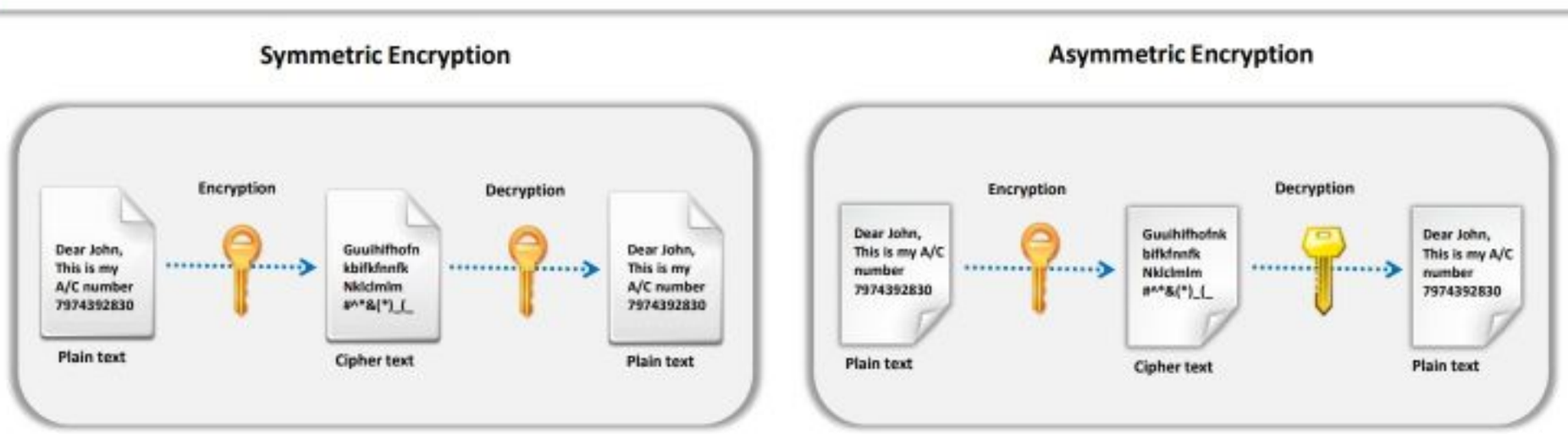


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Encryption: Symmetric and Asymmetric Key

- Encryption is a method of transforming plain text into something **unintelligible (cipher) text** in order to protect it from data compromise
- Symmetric encryption (secret-key, shared-key, and private-key) uses the same key for **encryption** as it does for **decryption**
- Asymmetric encryption (public-key) uses different encryption keys for encryption and decryption. These keys are known as **public** and **private keys**
- In Java, a cipher object is created in the process of encryption and decryption with a specific algorithm such as **DES** for symmetric and **RSA** for asymmetric encryption
- The **javax.crypto** package provides a framework for symmetric and asymmetric encryption with cipher implementations



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Encryption/Decryption Implementation Methods

- **Implementation methods for encryption and decryption**

1. Symmetric Key Generation

```
Basic.java
import java.security.NoSuchAlgorithmException;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

public class Basic {


private SecretKey generateKey ()
    throws NoSuchAlgorithmException {
    KeyGenerator key=KeyGenerator.getInstance("DESede");
return key.generateKey();
}
```

2. Encryption

```
private byte[] encrypt(byte[] message) throws
InvalidKeyException,
IllegalBlockSizeException,
BadPaddingException,
NoSuchAlgorithmException,
NoSuchPaddingException {

    Key aesKey = new SecretKeySpec(message, "AES");
Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, aesKey);
    return cipher.doFinal(message);
```

3. Decryption

```
byte[] encrypt(byte[] message) throws
InvalidKeyException,
IllegalBlockSizeException,
BadPaddingException,
NoSuchAlgorithmException,
NoSuchPaddingException {

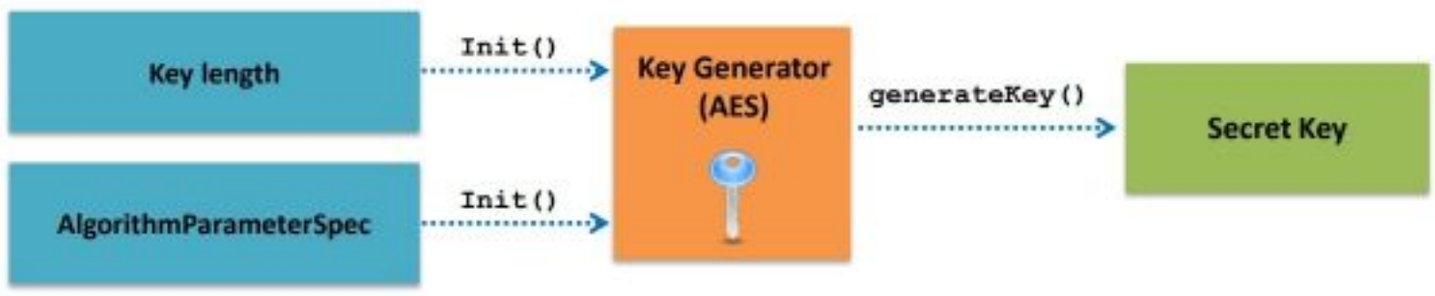
    Key aesKey = new SecretKeySpec(message, "AES");
Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.DECRYPT_MODE, aesKey);
    return cipher.doFinal(message);
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

SecretKeys and KeyGenerator

- SecretKeys are used in **symmetric encryption algorithms** and use the same keys for the encryption and decryption process
- KeyGenerator class creates the SecretKeys objects used by **cipher** for encryption and decryption
- KeyGenerator class `(javax.crypto.KeyGenerator);` is used to create **SecretKeys**
  - `SecretKey myKey = keyGen.generateKey();`



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Implementation Methods of KeyGenerator Class

```
public abstract class KeyGeneratorSpi
```

- KeyGenerator class abstracts with service provider interface class

```
protected abstract SecretKey engineGenerateKey()
```

- Generates secret keys using installed random number generator and installed algorithm parameter specification

```
protected abstract void engineInit(SecureRandom secRan)
```

- Initializes key generation engine with its generated random number

```
protected abstract void engineInit(AlgorithmParameterSpecaps,
  SecureRandomsecRan)
```

```
public final void engineInit(int strength, SecureRandom secRan)
```

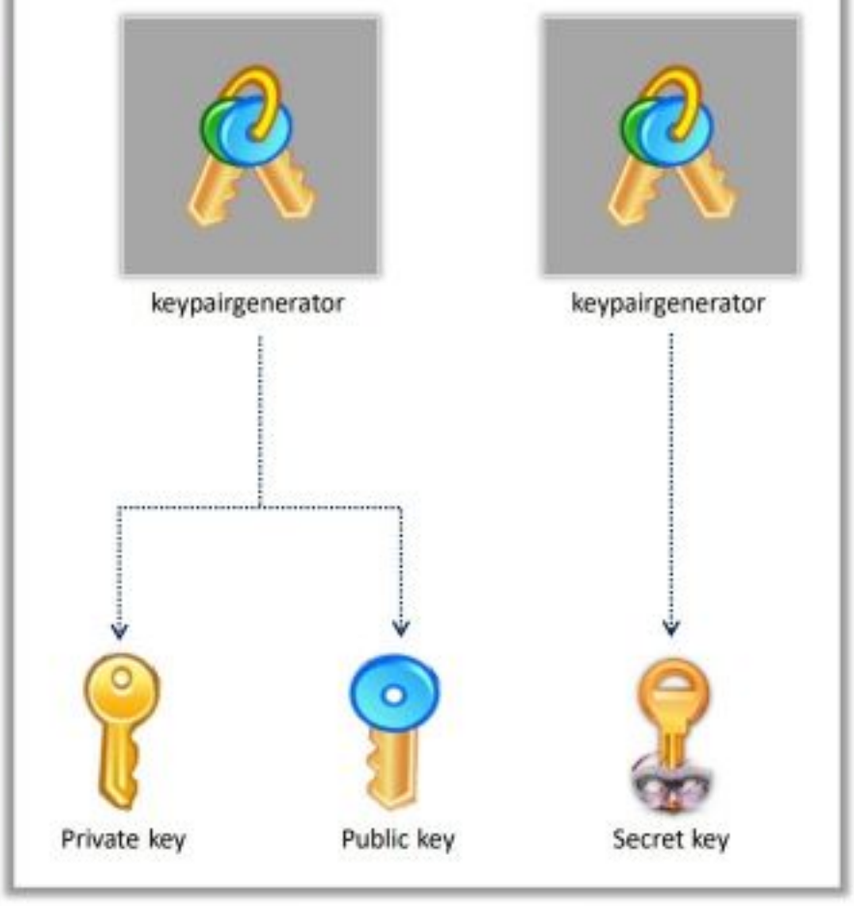
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Creating SecretKey with KeyGenerator Class

- In the code, **KeyPairGenerator** class is used for creating public and private keys

```
File Edit Refactor Source Navigate Search Project Run Window Help
Quick Access
KeyPairGenerator.java
1  package cryptography;
2
3  import java.security.Key;
4  import java.security.KeyPair;
5  import java.security.PublicKey;
6  import java.security.PrivateKey;
7
8  public class KeyPairGenerator {
9
10 private static String formatKey(Key key)
11 {
12 StringBuffer sbuf = new StringBuffer();
13 String alg = key.getAlgorithm();
14 String frmt = key.getFormat();
15 byte[] kEncoded = key.getEncoded();
16 sbuf.append("Key[algorithm=" + alg + ", format=" + frmt +
17 ", bytes=" + kEncoded.length + "]\n");
18 if (frmt.equalsIgnoreCase("RAW")){
19 sbuf.append("Key Material (in hex):: ");
20 sbuf.append(Util.byteArray2Hex(key.getEncoded()));
21  }
22 return sbuf.toString();
23 }
24
25 public static void main(String[] unused) throws Exception {
26  java.security.KeyPairGenerator key = java.security.KeyPairGenerator.getInstance("DSA");
27 ((java.security.KeyPairGenerator) key).initialize(512); // 512 is the keysize.
28 KeyPair pair = key.generateKeyPair();
29 PublicKey pubKey = pair.getPublic();
30 PrivateKey priKey= pair.getPrivate();
31 System.out.println("Generated Public Key:: " + formatKey(pubKey ));
32 System.out.println("Generated Private Key:: " + formatKey(priKey));
33 } }
```



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Creating SecretKey with KeyGenerator Class (Cont'd)

- In the code, **KeyGenerator** class is used for creating secret keys

```
File Edit Refactor Source Navigate Search Project Run Window Help
Quick Access
KeyPairGenerator.java
6
7  import javax.crypto.KeyGenerator;
8  import javax.crypto.SecretKey;
9  public class KeyPairGenerator {
10
11 private static String formatKey(Key key)
12 {
13 StringBuffer sbuf = new StringBuffer();
14 String alg = key.getAlgorithm();
15 String frmt = key.getFormat();
16 byte[] kEncoded = key.getEncoded();
17 sbuf.append("Key[algorithm=" + alg + ", format=" + frmt +
18 ", bytes=" + kEncoded.length + "]\n");
19 if (frmt.equalsIgnoreCase("RAW")){
20 sbuf.append("Key Material (in hex):: ");
21 sbuf.append(Util.byteArray2Hex(key.getEncoded()));
22  }
23 return sbuf.toString();
24 }
25
26 public static void main(String[] unused) throws Exception {
27
28    KeyGenerator keyGen = KeyGenerator.getInstance("DES");
29    keyGen.init(56); // 56 is the keysize. Fixed for DES
30    SecretKey skey = keyGen.generateKey();
31     System.out.println("Generated Key:: " + formatKey(skey));
32
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

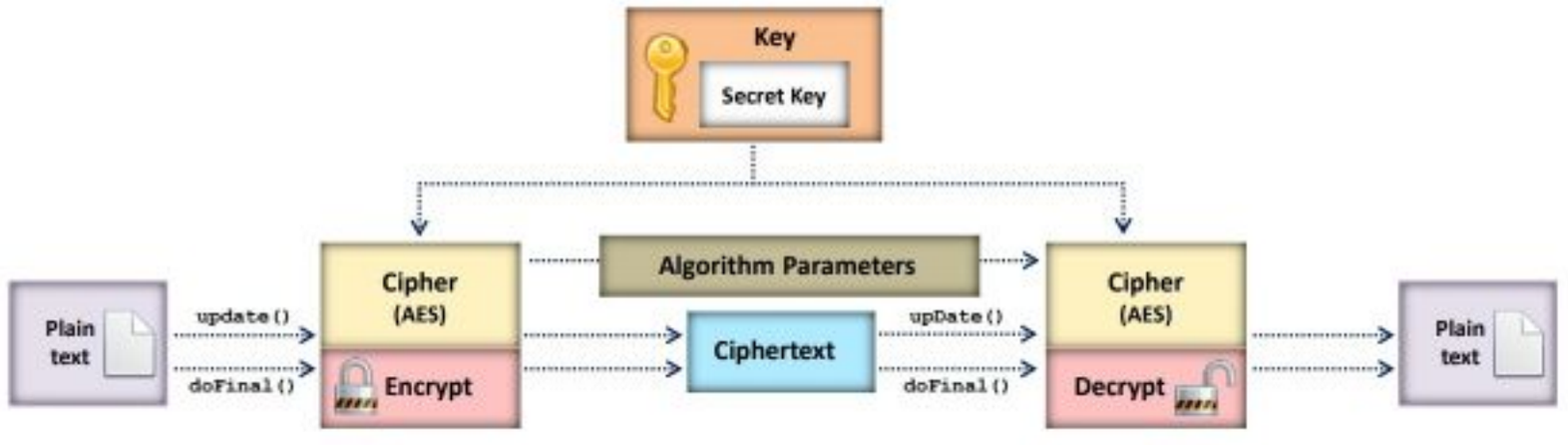
Cipher Class

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

The Cipher Class

- The Cipher class in `javax.crypto` package performs the main functionality of cryptographic cipher using encryption and decryption
- It is mostly implemented by programmers for **secure data transfers** by encrypting data from the sending end and decrypting data from the receiving end in order to safeguard data from intrusions while in transmission

Methods	Description
`getAlgorithm()`	Returns the name of the algorithm used by the cipher object
`getIV()`	Retrieves the initialization vector (IV) in a new buffer class
`getParameters()`	Retrieves the parameters used by the cipher
`doFinal()`	Performs encryption and decryption process as initialized by the cipher
`getBlockSize()`	Retrieves block size (bytes)



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Implementation Methods of Cipher Class

Description	Methods
Creating a cipher object	`public static Cipher getInstance(String transformation)` `public static Cipher getInstance(String transformation, String provider)`

Initializing a cipher object	`public void init(int opmode, Key key);`
	`public void init(int opmode, Certificate certificate)`
	`public void init(int opmode, Key key, SecureRandom random);`
	`public void init(int opmode, Certificate certificate, SecureRandom random`

Encrypting and Decrypting data	`public byte[] doFinal(byte[] input);` `public byte[] update(byte[] input);`
Wrapping and Unwrapping Keys	`public final byte[] wrap(Key key);` `public final Key unwrap(byte[] wrappedKey, String wrappedKeyAlgorithm, int wrappedKeyType));`

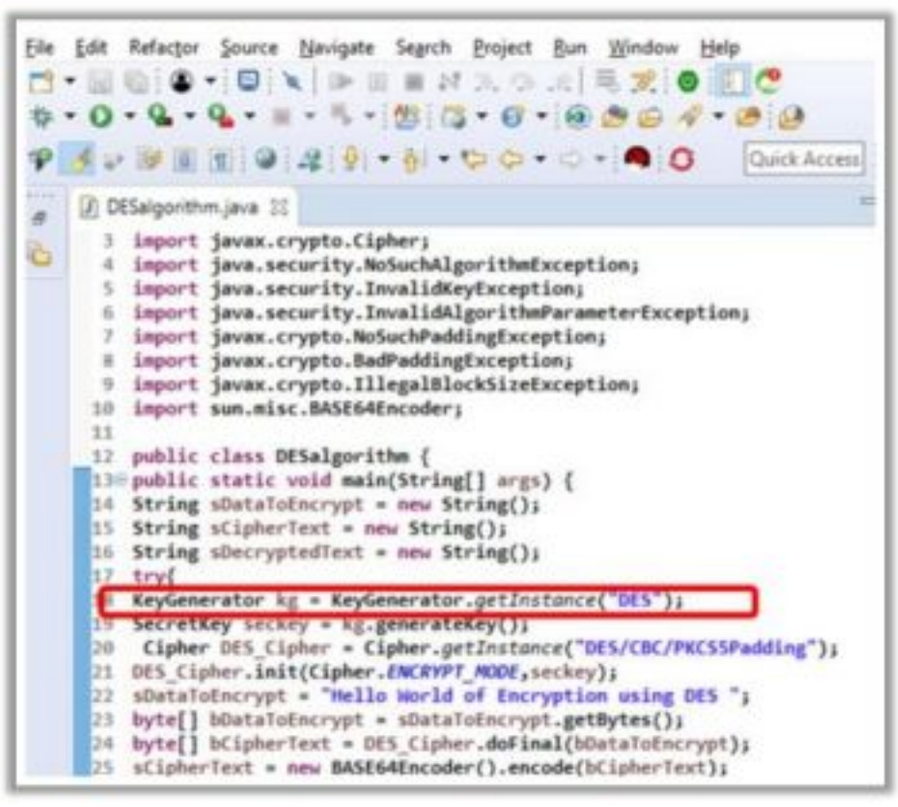
Managing Algorithm Parameters	`Init method`
	`Getparameters method`
	`getIV method`

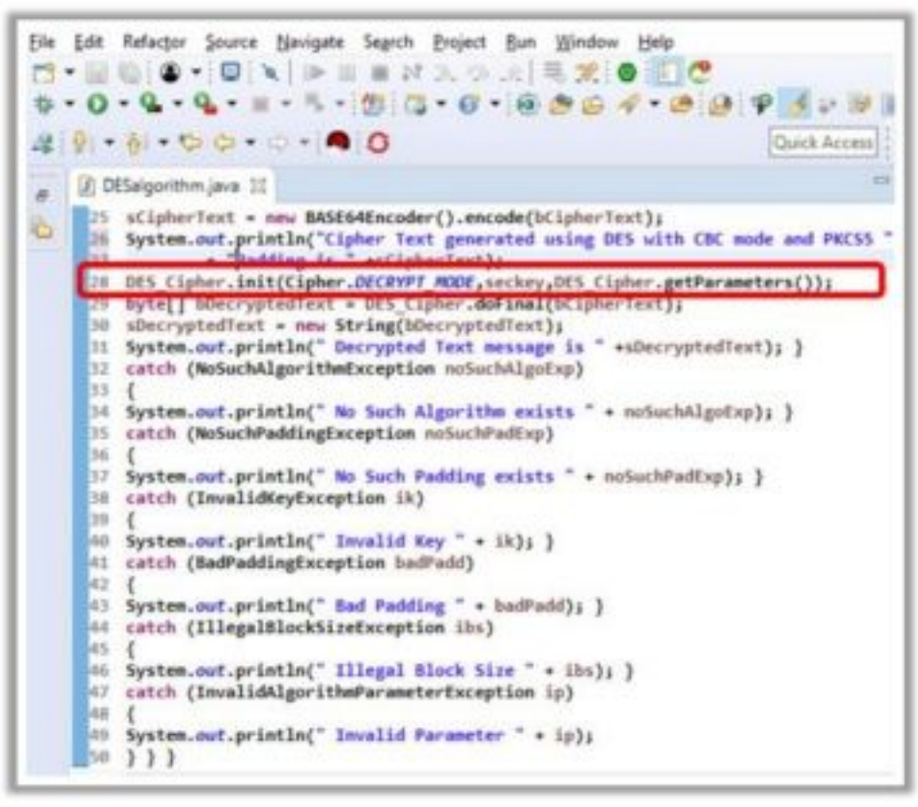
Cipher Output Considerations	`public int getOutputSize(int inputLen)`

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Insecure Code for Cipher Class using DES Algorithm

- The code uses the DES algorithm for creating Cipher class, although **DES** is a **standard algorithm** often used for constructing Cipher class, it is considered weak and prone to attacks





Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Code for Cipher Class using AES Algorithm

- The code uses the **AES** algorithm (considered as a strong algorithm according to OWASP) for creating **Cipher class** with **strong key size (128)**

```
1  package cryptography;
2
3  import javax.crypto.KeyGenerator;
4  import javax.crypto.SecretKey;
5  import javax.crypto.Cipher;
6  import java.security.NoSuchAlgorithmException;
7  import java.security.InvalidKeyException;
8  import java.security.InvalidAlgorithmParameterException;
9  import javax.crypto.NoSuchPaddingException;
10 import javax.crypto.BadPaddingException;
11 import javax.crypto.IllegalBlockSizeException;
12 import sun.misc.BASE64Encoder;
13 public class AESalgorithm {
14 public static void main(String[] args) {
15 String sDataToEncrypt = new String();
16 String sCipherText = new String();
17 String sDecryptedText = new String();
18 try{
19     KeyGenerator kg = KeyGenerator.getInstance("AES");
20     kg.init(128);
21     SecretKey seckey = kg.generateKey();
22     Cipher AES_Cipher = Cipher.getInstance("AES");
23     AES_Cipher.init(Cipher.ENCRYPT_MODE,seckey);
24     sDataToEncrypt = "Hello World of Encryption using AES ";
```

```
25     byte[] bDataToEncrypt = sDataToEncrypt.getBytes();
26     byte[] bCipherText = AES_Cipher.doFinal(bDataToEncrypt);
27     sCipherText = new BASE64Encoder().encode(bCipherText);
28     System.out.println("Cipher Text generated using AES is " +s
29     AES_Cipher.init(Cipher.DECRYPT_MODE,seckey,AES_Cipher.getPa
30     byte[] bDecryptedText = AES_Cipher.doFinal(bCipherText);
31     sDecryptedText = new String(bDecryptedText);
32     System.out.println(" Decrypted Text message is " +sDecrypte
33     }
34     catch (NoSuchAlgorithmException noSuchAlgoExp)
35     {
36     System.out.println(" No Such Algorithm exists " + noSuchAlg
37     catch (NoSuchPaddingException noSuchPadExp)
38     {
39     System.out.println(" No Such Padding exists " + noSuchPadEx
40     catch (InvalidKeyException ik)
41     {
42     System.out.println(" Invalid Key " + ik); }
43     catch (BadPaddingException badPadd)
44     {
45     System.out.println(" Bad Padding " + badPadd); }
46     catch (IllegalBlockSizeException ibs)
47     {
48     System.out.println(" Illegal Block Size " + ibs); }
49     catch (InvalidAlgorithmParameterException ip)
50     {
51     System.out.println(" Invalid Parameter " + ip);
52     } }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

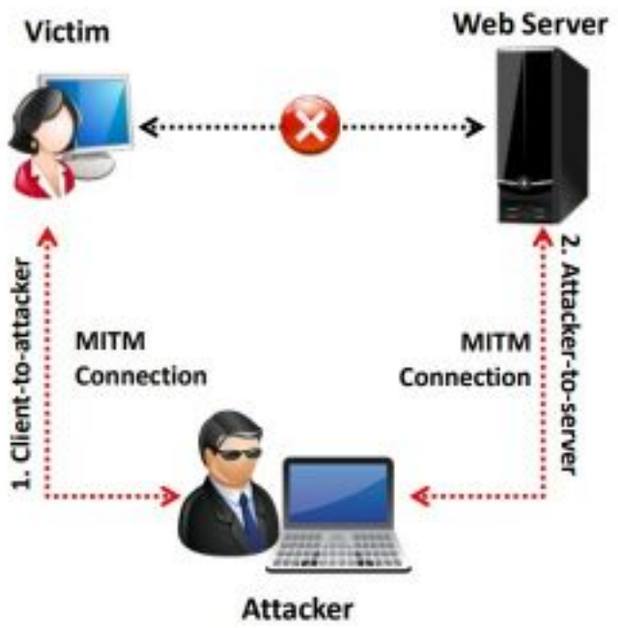
Digital Signatures

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Attack Scenario: Man-in-the-Middle Attack



- Man-in-the-Middle (MITM) attacks allow an attacker to access sensitive information by **intercepting** and **altering communications** between an end-user and webservers
- For any data that travels across the network, there is a possibility of manipulation or modification of data by intruders
- If the data is not adequately encrypted using digital signatures or digital certificates then there is a possibility of **data leakage**
- The information that travels may contain **personal data**, credit card numbers and passwords

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Digital Signatures

1. Digital signatures are used for protecting and maintaining the **integrity** of **digital data**, downloads, the source of the message and a software license file
2. JCA provides a specific framework for digital signatures with **java.security.Signature class** that provides the functionality of signing and verifying digital signatures
3. The Signature class and SignedObject class provide the basic functionality of **digitally signing** and **verifying** digital signatures in JCA



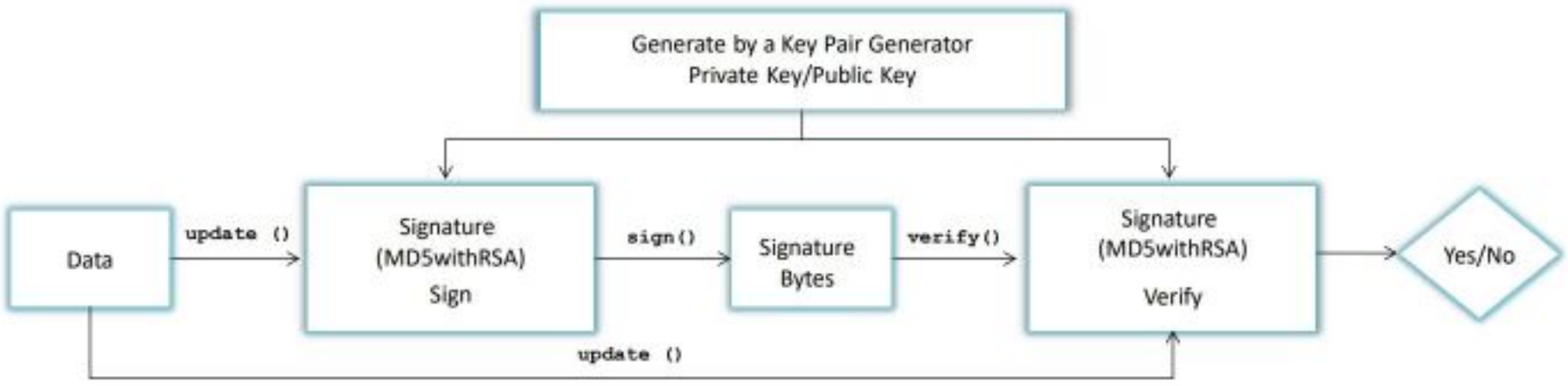
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

The Signature Class

- The Signature class is designed to provide applications for the functionality of a cryptographic digital signature algorithm such as **DSA** or **MD5withRSA**
- The Signature class can be used for checking whether a signature is in fact **authentic**
- Signature class methods include `getInstance( )`, `initSign( )`, `initVerify( )`, `update( )`, `sign( )`, and `verify( )`
- Only a private or public key owner can create a **signature**
- Public key for the desired private key is used to generate **signature**
- Public key and signature cannot reveal any details about **private key**



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Implementation Methods of Signature Class

Methods	Description
`Signature(String algorithm)`	Generates a signature object with a specific algorithm
`getAlgorithm()`	Returns the Signature object algorithm
`getInstance(String algorithm)`	Generates a signature object with a specific digest algorithm
`Getparameters()`	Returns the signature object's parameters
`sign()`	Retrieves signature bytes of data
`initSign(PrivateKey privateKey)`	Initializes the object for signing
`initVerify(Certificate certificate)`	Initializes the object for verification using a certificate
`initVerify(PublicKey publicKey)`	Initializes the object for verification
`update(byte b)`	Updates the data to be signed or verified by a byte

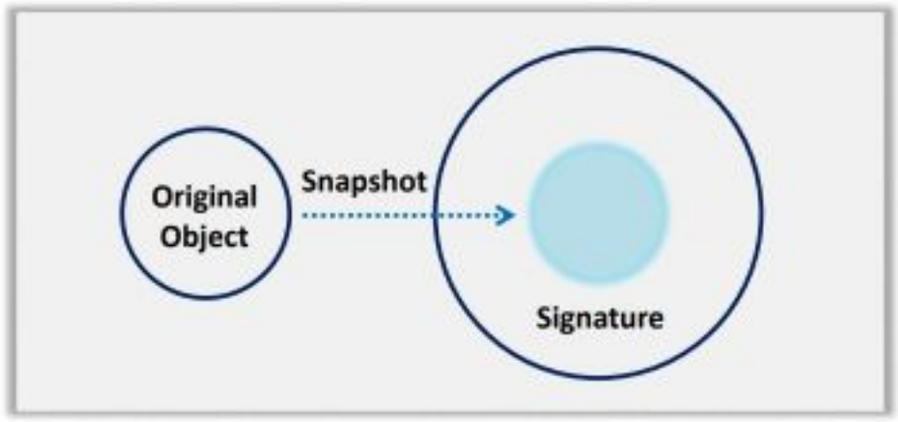
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

The SignedObjects

- Java provides developers with object level protection mechanisms, protecting the **integrity** and **confidentiality** of objects inside a runtime environment (memory) or that are in **transit** (stored in IP packets or saved on disk)
- **SignedObject** class is used for creating **authentic runtime objects** whose integrity cannot be compromised without being detected



- **SignedObject**

  (`java.security.SignedObject`):

  - Creates another serializable object (a copy of **serializable object**)
  - Creates in **serialized form**
  - Creates a signature using algorithms (**DSA**, **SHA-1**, **MD5**)
  - If the signature is not null then the signed object may contain a valid **digital signature**

- **Signedobjects are used as:**

  - Unforgetable **authorization tokens**
  - Sign and serialize **objects/data**
  - Verify authenticity of **digital signatures**
  - Construct a logical sequence of **signatures**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Implementing Methods of SignedObjects

Signing an object

```
Signed.java
19⊖   public void test() throws NoSuchAlgorithmException,
20    NoSuchProviderException, InvalidKeyException, SignatureException, IOException
21    {
22    Signature signingEngine = Signature.getInstance(algorithm, provider);
23    SignedObject sobj = new SignedObject(myobject, signingKey, signingEngine);
24
```

Verifying a SignedObject

```
*Signed.java
27    Signature verificationEngine = Signature.getInstance(algorithm, provider);
28    try {
29    if (sobj.verify(publickey, verificationEngine))
30    {
31        Object myob = sobj.getObject();
32    }}
33    catch (java.lang.ClassNotFoundException e) {
```

- **myobject:** The serialized object that needs to be signed
- **signingKey:** A private key such as DSAPrivateKey, RSAPrivateKey, RSAPrivateCrtKey
- **signingEngine:** Signature Algorithms can be SHA1withDSA, AES, etc.
- **publickey:** Key used for verification
- **verificationEngine:** Signature verification engine

Methods	Description
`getAlgorithm()`	Gets the name of the algorithm used to seal an object
`getObject()`	Gets the encapsulate object
`getSignature()`	Gets the signature of the signed object
`verify(PublicKey verificationKey, Signature verificationEngine)`	Verifies if a signedobject contains a valid signature

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

The SealedObjects

- Sealedobject class helps developers in creating an object and protecting its confidentiality using **cryptographic algorithms**

- Sealedobject (`javax.crypto.SealedObject`)
  - Aims at protecting object confidentiality
  - Creates serializable object
  - Creates cryptographic algorithm (DES, IDEA, RC4)
  - Encryption and decryption
  - Deserialization of original object

- The proper way of maintaining the integrity and confidentiality of **Java objects** that travel across network is to:
  - First, create **signedobject**
  - Second, create **sealedobject**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Implementation Methods of SealedObject

Encryption

Signed.java

```
57
58  KeyGenerator k = KeyGenerator.getInstance ("DES");
    SecretKey desKey = k.generateKey();
    Cipher c = Cipher.getInstance ("DES");
    c.init(Cipher.ENCRYPT_MODE, deSKEY);
    String str = new String ("Good Morning");
    SealedObject Sobj = new SealedObject (str, c);
62
63
```

Decryption

Signed.java

```
57
58  KeyGenerator k = KeyGenerator.getInstance ("DES");
    SecretKey desKey = k.generateKey();
    Cipher c = Cipher.getInstance ("DES");
    c.init(Cipher.DECRYPT_MODE, deSKEY);
    String str = new String ("Good Morning");
    SealedObject Sobj = new SealedObject (str, c);
62
63
```

Methods	Description
`getAlgorithm()`	Returns the name of the algorithm used to seal an object
`getObject(Cipher c)`	Retrieves the original (encapsulate) object
`getObject(Key key)`	Retrieves the original (encapsulate) object

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Insecure and Secure Code for Signed/Sealed Objects

Vulnerable Code

- In the given code, sealing is performed before signing and that questions the integrity of the signer's true identity




```
13 import java.security.Signature;
14 import java.security.SignedObject;
15 import javax.crypto.Cipher;
16 import javax.crypto.KeyGenerator;
17 import javax.crypto.SealedObject;
18
19 public class SecureSigned {
20     public static void main(String[] args) throws IOException,
21     GeneralSecurityException, ClassNotFoundException {
22     SerializableMap<String, Integer> map = buildMap();  // Build map
23     KeyGenerator KeyGen;    // Generate sealing key & seal map
24     KeyGen = KeyGenerator.getInstance("AES");
25     KeyGen.init(new SecureRandom());
26     Key k = KeyGen.generateKey();
27     Cipher cipher1 = Cipher.getInstance("AES");
28     cipher1.init(Cipher.ENCRYPT_MODE, k);
29     SealedObject sealedMap = new SealedObject(map, cipher1);
30     KeyPairGenerator keyPairGen =KeyPairGenerator.getInstance("DES");
31     KeyPair kypair = keyPairGen.generateKeyPair();
32     Signature sign = Signature.getInstance("SHA1withDSA");
33     SignedObject signedMap =new SignedObject(sealedMap, kypair.getPrivate(), sign);
34     ObjectOutputStream out =new ObjectOutputStream(new FileOutputStream("data")); // Seri
35     out.writeObject(signedMap); out.close();
36     ObjectInputStream ins =new ObjectInputStream(new FileInputStream("data"));  // Deseri
37     signedMap = (SignedObject) ins.readObject(); ins.close();
38     if (!signedMap.verify(kypair.getPublic(), sign)) {  // Verify signature and retrieve
39         throw new GeneralSecurityException("Map failed verification"); }
40         sealedMap = (SealedObject) signedMap.getObject();
41         cipher1 = Cipher.getInstance("AES"); // Unseal map
42         cipher1.init(Cipher.DECRYPT_MODE, k);
43         map = (SerializableMap<String, Integer>) sealedMap.getObject(cipher1);
44         InspectMap(map); // Inspect map
45         }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Insecure and Secure Code for Signed/Sealed Objects (Cont'd)

Secure Code

- In the given code, signing is performed before sealing the object and that guarantees the authenticity of the object




```
12 import java.security.SecureRandom;
13 import java.security.Signature;
14 import java.security.SignedObject;
15 import javax.crypto.Cipher;
16 import javax.crypto.KeyGenerator;
17 import javax.crypto.SealedObject;
18
19 public class SecureSigned {
20     public static void main(String[] args) throws IOException,
21     GeneralSecurityException, ClassNotFoundException {
22     SerializableMap<String, Integer> map = buildMap();  // Build map
23     KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");
24     KeyPair kypair = keyPairGen.generateKeyPair();
25     Signature sign = Signature.getInstance("SHA1withDSA");
26     SignedObject signedMap = new SignedObject(map, kypair.getPrivate(), sign);
27     KeyGenerator KeyGen;    // Generate sealing key & seal map
28     KeyGen = KeyGenerator.getInstance("AES");
29     KeyGen.init(new SecureRandom());
30     Key k = KeyGen.generateKey();
31     Cipher cipher1 = Cipher.getInstance("AES");
32     cipher1.init(Cipher.ENCRYPT_MODE, k);
33     SealedObject sealedMap = new SealedObject(map, cipher1);
34     // Generate signing public/private key pair & sign map
35     ObjectOutputStream out =new ObjectOutputStream(new FileOutputStream("data"));
36     out.writeObject(signedMap); out.close();
37     ObjectInputStream ins =new ObjectInputStream(new FileInputStream("data"));
38     signedMap = (SignedObject) ins.readObject(); ins.close();
39     if (!signedMap.verify(kypair.getPublic(), sign)) {  // Verify signature and
40         throw new GeneralSecurityException("Map failed verification"); }
41         sealedMap = (SealedObject) signedMap.getObject();
42         cipher1 = Cipher.getInstance("AES"); // Unseal map
43         cipher1.init(Cipher.DECRYPT_MODE, k);
44         map = (SerializableMap<String, Integer>) sealedMap.getObject(cipher1);
45         InspectMap(map); // Inspect map
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Java XML Digital Signature

- XML Signature syntax, processing rules and representing are defined by W3C XML Signature Working Group and W3C XML Security Specifications Maintenance Working Group
- Java XML Digital Signature API Specification (JSR 105) is to define a standard Java API for generating and validating XML signatures. The API are located in javax.xml.crypto package
- The purposes are to provide:
  - XML Canonicalization
  - Signature Generation
    - Detached XML Digital Signature
    - Enveloped XML Digital Signature
    - Enveloping XML Digital Signature
  - Signature Validation

Example: XML Digital Signature

sig.xml

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n -20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>...<DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue></SignatureValue>
  <KeyInfo>
    <KeyValue>
      <DSAKeyValue>
       <P></P><Q></Q><G></G><Y></Y>
       </DSAKeyValue>
    </KeyValue>
  </KeyInfo>
</Signature>
```

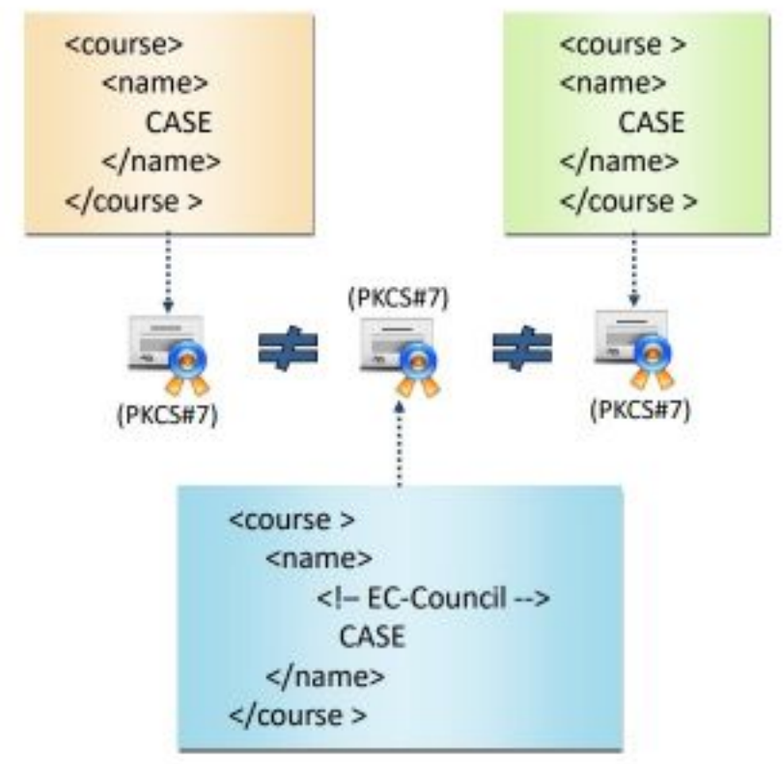
Design Source

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java XML Digital Signature (Cont'd)

XML Canonicalization

- **Digital signature (PKCS#7)** only works if the verification calculations are performed on exactly the same bits as the signing calculations, which is inappropriate for XML data
- XML data is subject to surface representation changes which needs to be canonicalized to DOM-Level, character encoding and XML namespace while calculating the signature



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Java XML Digital Signature (Cont'd)

Detached XML Digital Signature

- The signature is detached from the content it signs. It applies to separate data objects and also the signature and data object reside within the same XML document as sibling elements




```
*Myclass.java
40    //create a DOM XMLSignatureFactory
41
42    //create a DOM XMLSignatureFactory
43    XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM");
44    //using the SHA1 digest algorithm
45    Reference ref = (Reference) fac.newReference("http://www.w3.org/TR/xmlstylesheet",
46            fac.newDigestMethod(DigestMethod.SHA1, null));
47    // Create the SignedInfo
48    SignedInfo si = fac.newSignedInfo(fac.newCanonicalizationMethod
49              (CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS,
50              (C14NMethodParameterSpec) null),
51              fac.newSignatureMethod(SignatureMethod.DSA_SHA1, null),
52              Collections.singletonList(ref));
53    // Create a DSA KeyPair
54    KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
55    kpg.initialize(512);
56    KeyPair kp = kpg.generateKeyPair();
57    // Create DSA public key KeyValue object
58    KeyInfoFactory kif = fac.getKeyInfoFactory();
59    KeyValue kv = kif.newKeyValue(kp.getPublic());
60    // Create a KeyInfo and add the KeyValue to it
61    KeyInfo ki = kif.newKeyInfo(Collections.singletonList(kv));
62    // Create the XMLSignature object
63    XMLSignature signature = fac.newXMLSignature(si, ki);
64    //Create the Document that will hold the resulting XMLSignature
65    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
66    dbf.setNamespaceAware(true);
67    Document doc = dbf.newDocumentBuilder().newDocument();
68    // Create a DOMSignContext and set the signing Key
69    DOMSignContext signContext = new DOMSignContext(kp.getPrivate(), doc);
70    // Marshal, generate (and sign) the detached XMLSignature
71    signature.sign(signContext);
72    // output the resulting document
73    OutputStream os;
74    if (args.length > 0) {
75       os = new FileOutputStream(args[0]);
76    } else {
77       os = System.out;
78    }
79    TransformerFactory tf = TransformerFactory.newInstance();
80    Transformer trans = tf.newTransformer();
81    trans.transform(new DOMSource(doc), new StreamResult(os));
82
83
84 }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java XML Digital Signature (Cont'd)

Enveloping XML Digital Signature

- The signature is over the content found within an object element of the signature itself




```
Myclass.java
42
43
44  private void main(String args[]) throws NoSuchAlgorithmException, InvalidAlgorithmParame
45      //create a DOM XMLSignatureFactory
46
47      XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM");
48      Reference ref = (Reference) fac.newReference("#object",
49         fac.newDigestMethod(DigestMethod.SHA1, null));
50      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
51      dbf.setNamespaceAware(true);
52      Document doc = dbf.newDocumentBuilder().newDocument();
53      Node text = doc.createTextNode("some text");
54      XMLStructure content = new DOMStructure(text);
55      XMLObject obj = fac.newXMLObject
56              (Collections.singletonList(content), "object", null, null);
57      // Create the SignedInfo
58      SignedInfo si = fac.newSignedInfo(fac.newCanonicalizationMethod
59                (CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS,
60                (C14NMethodParameterSpec) null),
61                fac.newSignatureMethod(SignatureMethod.DSA_SHA1, null),
62                Collections.singletonList(ref));
63      // Create a DSA KeyPair
64      KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
65      kpg.initialize(512);
66      KeyPair kp = kpg.generateKeyPair();
67      // Create DSA public key KeyValue object
68      KeyInfoFactory kif = fac.getKeyInfoFactory();
69      KeyValue kv = kif.newKeyValue(kp.getPublic());
70      // Create a KeyInfo and add the KeyValue to it
71      KeyInfo ki = kif.newKeyInfo(Collections.singletonList(kv));
72      // Create the XMLSignature object
73      XMLSignature signature = fac.newXMLSignature(si, ki,
74                        Collections.singletonList(obj), null, null);
75      // Create a DOMSignContext and specify the DSA PrivateKey
76      DOMSignContext dsc = new DOMSignContext(kp.getPrivate(), doc);
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Java XML Digital Signature (Cont'd)

Enveloped XML Digital Signature

- The signature is over the XML content that contains the signature as an element. The content provides the root XML document element




```
<Envelope>
  <data>...</data>
  <Signature>...</Signature>
</ Envelope >
```

Myclass.java

```
47
48
49  private void main(String args[]) throws NoSuchAlgorithmException, InvalidAlgorithmParam
50      //create a DOM XMLSignatureFactory
51
52      XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM");
53      Reference ref = (Reference) fac.newReference("",
54         fac.newDigestMethod(DigestMethod.SHA1, null),
55         Collections.singletonList
56           (fac.newTransform
57      (Transform.ENVELOPED, (TransformParameterSpec) null)), null, null);
58      // Create the SignedInfo
59      SignedInfo si = fac.newSignedInfo(fac.newCanonicalizationMethod
60                (CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS,
61                (C14NMethodParameterSpec) null),
62                fac.newSignatureMethod(SignatureMethod.DSA_SHA1, null),
63                Collections.singletonList(ref));
64      // Create a DSA KeyPair
65      KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
66      kpg.initialize(512);
67      KeyPair kp = kpg.generateKeyPair();
68      // Create DSA public key KeyValue object
69      KeyInfoFactory kif = fac.getKeyInfoFactory();
70      KeyValue kv = kif.newKeyValue(kp.getPublic());
71      // Create a KeyInfo and add the KeyValue to it
72      KeyInfo ki = kif.newKeyInfo(Collections.singletonList(kv));
73      // Instantiate the document to be signed
74      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
75      dbf.setNamespaceAware(true);
76      Document doc =    dbf.newDocumentBuilder().parse(new FileInputStream(""));
77      // Create a DOMSignContext and specify the DSA PrivateKey
78      DOMSignContext dsc = new DOMSignContext
79              (kp.getPrivate(), doc.getDocumentElement());
80      // Create the XMLSignature object
81      XMLSignature signature = fac.newXMLSignature(si, ki);
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java XML Digital Signature (Cont'd)

Validate XML Digital Signature

- The validation process includes signature validation and SignedInfo reference validation



Myclass.java | KeyValueKeySelector.java

```
55
56      // Instantiate the document to be validated
57      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
58      dbf.setNamespaceAware(true);
59      Document doc =
60           dbf.newDocumentBuilder().parse(new FileInputStream(""));
61      // Find Signature element
62      NodeList nl =
63         doc.getElementsByTagNameNS(XMLSignature.XMLNS, "Signature");
64      if (nl.getLength() == 0) {
65          throw new Exception("Cannot find Signature element");})
66      // Create a DOM XMLSignatureFactory
67      XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM");
68      // Create a DOMValidateContext and specify a KeyValue KeySelector
69      DOMValidateContext valContext = new DOMValidateContext
70      (new KeyValueKeySelector(), nl.item(0));
71      // unmarshal the XMLSignature
72      XMLSignature signature = fac.unmarshalXMLSignature(valContext);
73      // Validate the XMLSignature (generated above)
74      boolean coreValidity = signature.validate(valContext);
75      // Check core validation status
76      if (coreValidity == false) {
77         System.err.println("Signature failed core validation");
78         boolean sv = signature.getSignatureValue().validate(valContext);
79         System.out.println("signature validation status: " + sv);
80         // check the validation status of each Reference
81         Iterator i =  signature.getSignedInfo().getReferences().iterator();
82         for (int j=0; i.hasNext(); j++) {
83            boolean refValid =
84                ((XMLSignature) i.next()).validate(valContext);
85           System.out.println("ref["+j+"] validity status: " + refValid);
86         }
87      } else {
88          System.out.println("Signature passed core validation");
89      }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Socket Layer (SSL)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secure Socket Layer (SSL)

- **Data** that has traveled across the **network** cannot be trusted; it can be intercepted intentionally or unintentionally and used for unauthorized purposes
- Secure Sockets Layer (SSL) is a widely used application layer **cryptographic protocol** that secures communication over the internet through:
  - **Encryption**

    Protects data in transmission from intrusions with **encrypt** and **decrypt methods**
  - **Authentication**

    Verifies sender's identity by providing certificates and securing connections between **client** and **server**
  - **Digital Certificates**

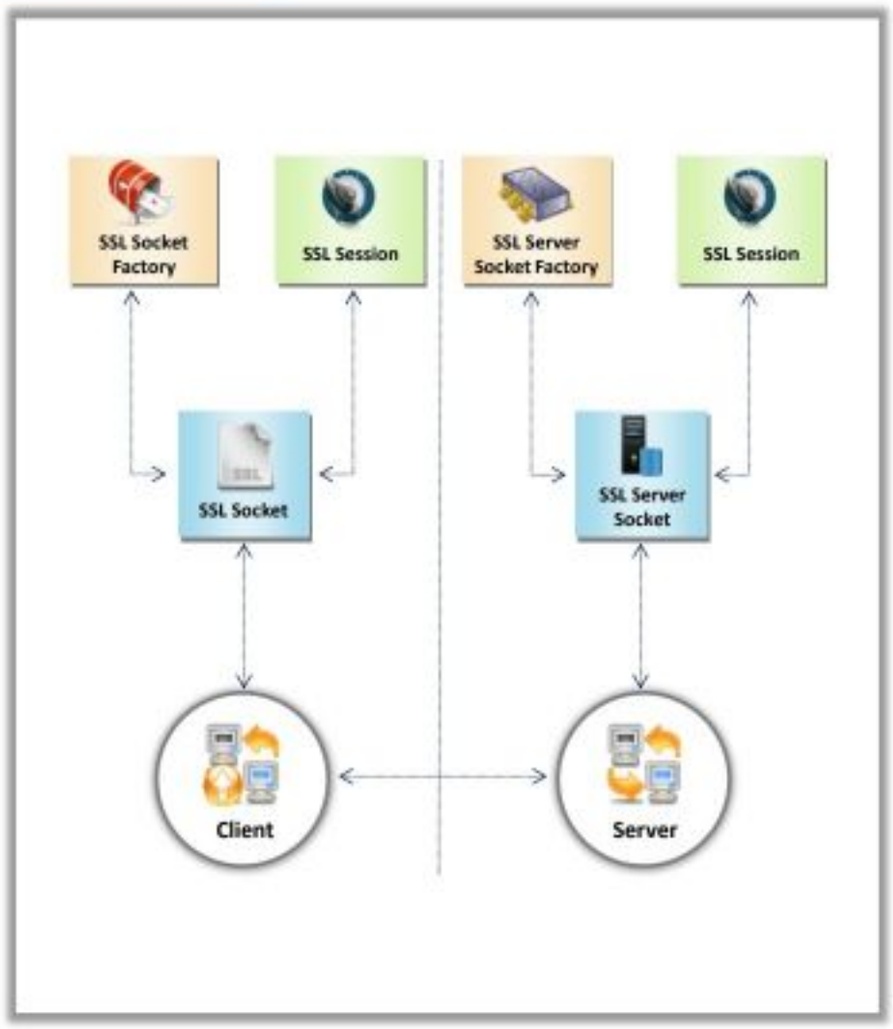
    Validates the certificate issued by **CA** ensuring the integrity of data in transmission

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Java Secure Socket Extension (JSSE)

- JSSE provides API framework and its **implementation** of the **SSL** and **TLS protocols** imbibed with functionalities such as encryption, server authentication, client authentication and message integrity
- It also provides support for **HTTPS** and several **cryptographic algorithms** that minimize subtle and dangerous security vulnerabilities
- `javax.net.ssl.SSLSocket` class supports standard socket methods to secure socket communication
- `javax.net` and `javax.net.ssl` packages are the standard JSSE APIs that include important classes such as:
  - SSLServerSocket
  - SSLServerSocketFactory
  - SSLSocket
  - SocketFactory
  - ServerSocketFactory
  - SSLSocketFactory classes

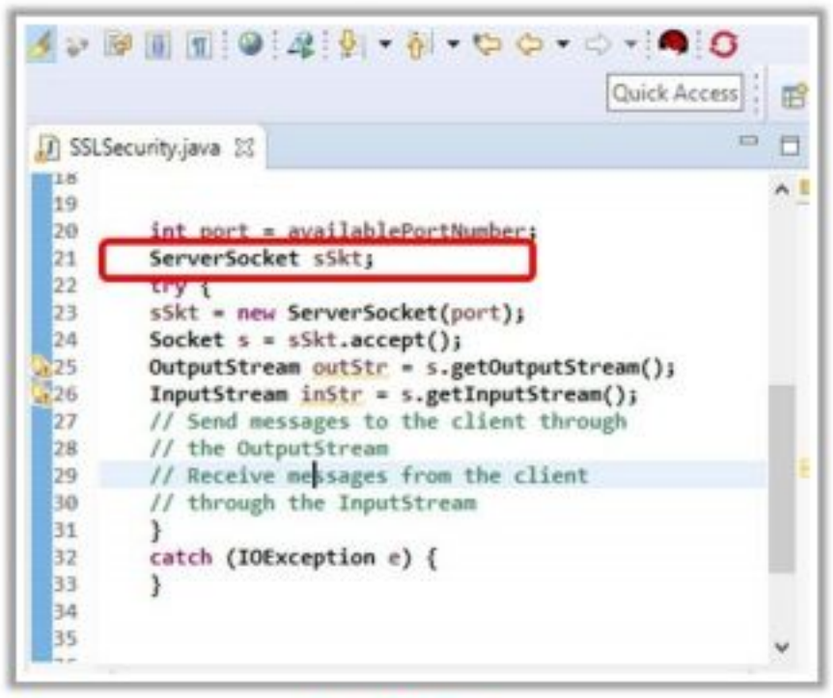


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SSL and Security: Example 1

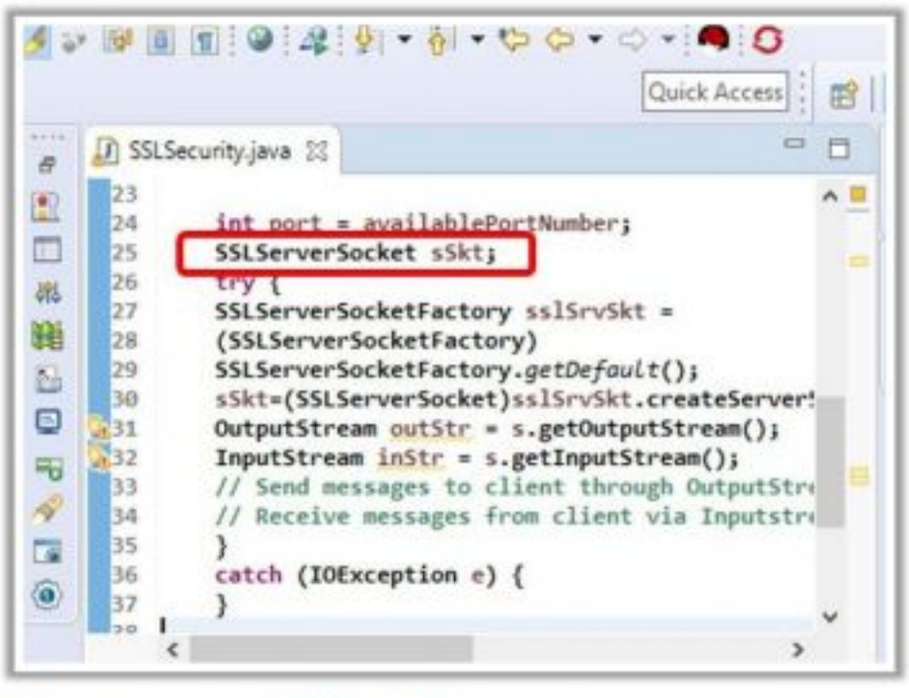
- Sockets are used as a communication channel between the server and client in **Java applications**

Vulnerable Code



- The server code uses **regular sockets** that fail to protect data in transmission

Secure Code



- The server code uses **secure sockets** to protect data while in transmission

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

SSL and Security: Example 2

Vulnerable Code

- The code uses **regular sockets** for a server application that fails to protect data in transmission

```
6  import java.io.*;
7  import java.net.*;
8  public class SSLSecurity {
9
10     public void EcoServer(String[] args) throws IOException {
11     ServerSocket SrvSkt = null;
12     try {
13     SrvSkt = new ServerSocket(9999);
14     Socket s = SrvSkt.accept();
15     PrintWriter out = new PrintWriter(s.getOutputStream(), true);
16     BufferedReader br = new BufferedReader(
17     new InputStreamReader(s.getInputStream()));
18     String inLine;
19     while ((inLine = br.readLine()) != null) {
20     System.out.println(inLine);
21     out.println(inLine);
22     }
23     } finally {
24     if (SrvSkt != null) {
25     try {
26     SrvSkt.close();
27     } catch (IOException x) {
28     // handle error
29     } } }     }
30     public void EcoClient(String[] args) throws IOException {
31         Socket s = null;
32         try {
33         s = new Socket("localhost", 9999);
34         PrintWriter out = new PrintWriter(s.getOutputStream(), tru
35         BufferedReader br = new BufferedReader(
36         new InputStreamReader(s.getInputStream()));
37         BufferedReader In = new BufferedReader(
38         new InputStreamReader(System.in));
39         String user_input;
40         while ((user_input = In.readLine()) != null) {
41         out.println(user_input);
42         System.out.println(br.readLine());
43         }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SSL and Security: Example 2 (Cont'd)

Secure Code

- The code uses **SSLSocket** that blocks any attempts to connect any ports that are not using SSL

Demo.java

```
1  package com.demo;
2
3  import java.io.IOException;
4  import javax.net.ssl.SSLServerSocket;
5  import javax.net.ssl.SSLServerSocketFactory;
6  import javax.net.ssl.SSLSocket;
7  import java.io.PrintWriter;
8  import java.io.BufferedReader;
9  import java.io.InputStreamReader;
10 public class Demo{
11
12     public static void main(String[] args) throws IOException {
13         SSLServerSocket sslSrvSkt = null;
14         try {
15         SSLServerSocketFactory sslSrvSktFactory =
16         (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
17         sslSrvSkt = (SSLServerSocket) sslSrvSktFactory.
18         createServerSocket(9999);
19         SSLSocket sslSkt = (SSLSocket) sslSrvSkt.accept();
20         PrintWriter out = new PrintWriter(sslSkt.getOutputStream(),true);
21         BufferedReader br = new BufferedReader(
22         new InputStreamReader(sslSkt.getInputStream()));
23         String inLine;
24         while ((inLine = br.readLine()) != null) {
25         System.out.println(inLine);
26         out.println(inLine); }
27         } finally {
28 if (sslSrvSkt != null) {
29 try {
30 sslSrvSkt.close();
31 } catch (IOException x) {
32 // handle error
33 }
```

Demo.java

```
41     public void EchoClient() throws UnknownHostException, IOException
42     {
43
44         SSLSocket sslSkt = null;
45         try {
46         SSLSocketFactory sslSktFactory =
47         (SSLSocketFactory) SSLSocketFactory.getDefault();
48         sslSkt =
49         (SSLSocket) sslSktFactory.createSocket("localhost", 9999);
50         PrintWriter out = new PrintWriter(sslSkt.getOutputStream(), true);
51         BufferedReader br = new BufferedReader(
52         new InputStreamReader(sslSkt.getInputStream()));
53         BufferedReader In = new BufferedReader(
54         new InputStreamReader(System.in));
55         String user_input;
56         while ((user_input = In.readLine()) != null) {
57         out.println(user_input);
58         System.out.println(br.readLine()); }
59         } finally {
60         if (sslSkt != null) {
61         try {
62         sslSkt.close();
```

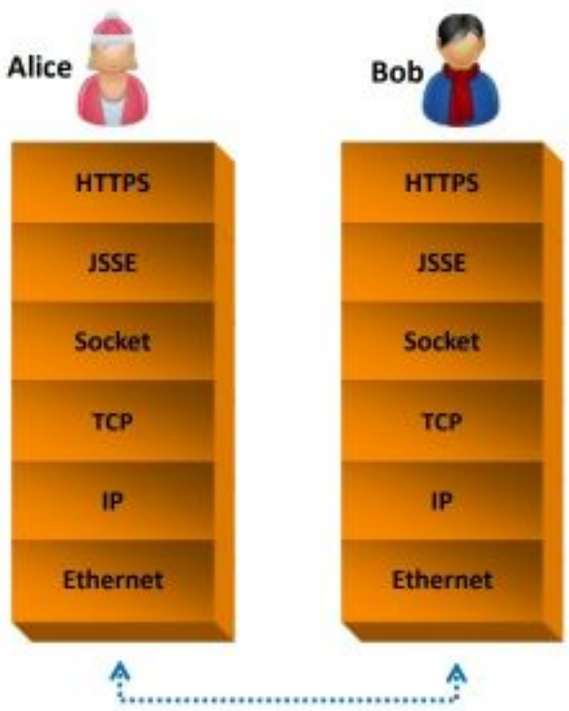
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

JSSE and HTTPS

- HTTP and HTTPS are **request** and **response** protocols where a client sends a request and a server sends a response to the client
- Some methods supported by these protocols include:
  - `GET`
  - `POST`
  - `DELETE`
  - `PUT`
- HTTPS protocol:
  - Creates a secure connection via SSL/TLS sockets
  - Verifies sender's identity by checking certificates
  - Performs requesting/receiving of data
- `javax.net.ssl.HttpsURLConnection` is an extension of `java.net.HttpsURLConnection` class supporting https-specific features
- Classes used for **HTTPS URL connections** include
  - `java.net.URL, java.net.URLConnection,`
  - `java.net.HttpURLConnection, and`
  - `javax.net.ssl.HttpURLConnection`



JSSE encapsulates sockets and TCP/IP layers Cross-Reference

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

JSSE and HTTPS (Cont'd)









Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Insecure HTTP Server Code

- In the code, **ProcessConnection** class is used for creating a new request in **multi-threaded HTTP server** and **shipDocument** method to send a document to the sever; however, the authenticity of the communication is questioned

InsecureHttp.java

```
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.net.ServerSocket;
8  import java.net.Socket;
9
10 public class InsecureHttp {
11     // The port number which the server will be listening on 44
12        public static final int HTTP_PORT = 8080;
13     public ServerSocket getServer() throws Exception {
14     return new ServerSocket(HTTP_PORT);
15        }
16     // multi-threading -- create a new connection for each request
17     public void run() {
18     ServerSocket hear;
19     try {
20     hear = getServer(); while(true) {
21     Socket sClient = hear.accept();
22     ProcessConnection conn = new ProcessConnection(sClient);
23           }
24           } catch(Exception e) {
25              System.out.println("Exception: "+e.getMessage());
26           } }
27          // main program
28            public static void main(String argv[]) throws
29           Exception {
30          HttpServer httpsrv = new HttpServer(); httpsrv.run();
31            } }
32          class ProcessConnection extends Thread {
33          Socket sClient; BufferedReader bIs; DataOutputStream bOs;
34          public ProcessConnection(Socket skt) { // constructor
35           sClient = skt;
36           try {
37           bIs = new BufferedReader(new InputStreamReader
38           (sClient.getInputStream()));
39           bOs = new DataOutputStream(sClient.getOutputStream());
40                } catch (IOException e) {
41           System.out.println("Exception: "+e.getMessage());
42           }    this.start(); // Thread starts here...this start()
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Insecure HTTP Server Code (Cont'd)

InsecureHttp.java

```
50
51          public void run() {
52              try {
53          // get a request and parse it.
54          String req = bIs.readLine(); System.out.println( "Request: "+req );
55          StringTokenizer sToken = new StringTokenizer( req );
56          if ((sToken.countTokens() >= 2) &&
57          sToken.nextToken().equals("GET") ) {
58          if ((req = sToken.nextToken()).startsWith("/") ) req = req.substring( 1 );
59          if ( req.equals("") )
60          req = req + "index.html";
61          File f1 = new File(req); shipDocument(bOs, f1);
62          } else {
63          bOs.writeBytes( "400 Bad Request" );
64          }
65          sClient.close(); } catch (Exception e) {
66          System.out.println("Exception: " + e.getMessage());
67            }
68             }
69            /**
70             * Read the requested file and ships it to the browser if found.*/
71
72
73          public static void shipDocument(DataOutputStream outs,
74                  File f1) throws Exception {
75                  try {
76                  DataInputStream ins = new DataInputStream(new FileInputStream(f1));
77                  int length = (int) f1.length(); byte[] buffer = new byte[length];
78                  ins.readFully(buffer); ins.close();
79                  outs.writeBytes("HTTP/1.0 200 OK\r\n");
80                  outs.writeBytes("Content-Length: " + f1.length() +"\r\n");
81                  outs.writeBytes("Content-Type:text/html\r\n\r\n"); outs.write(buffer); outs.flush();
82                  } catch (Exception e) {
83                  outs.writeBytes("<html><head><title>error</title></head><body>\r\n\r\n");
84                  outs.writeBytes("HTTP/1.0 400 " + e.getMessage() + "\r\n");
85                  outs.writeBytes("Content-Type: text/html\r\n\r\n");
86                  outs.writeBytes("</body></html>"); outs.flush();
87                  } finally {
88                  outs.close();
89                        }
90                   }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure HTTP Server Code

- In the code, before creating a connection with the HTTP server, a certificate is generated to make it more secure and it uses the **getServer()** method to make a secure server socket connection on HTTPS default port number 443

HttpSecure.java | HttpsServer.java

```
1  package com.domo;
2
3  import java.io.FileInputStream;
4  import java.net.ServerSocket;
5  import java.net.Socket;
6  import java.security.KeyStore;
7  import javax.net.ServerSocketFactory;
8  import javax.net.ssl.KeyManagerFactory;
9  import javax.net.ssl.SSLContext;
10 import javax.net.ssl.SSLServerSocket;
11
12 public class HttpSecure {
13     /**
14      * This class implements a multithreaded simple HTTPS
15      * server that supports the GET request method.
16      * It listens on port 44, waits client requests
17      * and serves documents.     */
18       String key_store = "serverkeys";
19       char key_storepass[] = "hellothere".toCharArray();
20       char keyPasswd[] = "hiagain".toCharArray();
21       // The port number which the server will be listening on
22       public static final int HTTPS_PORT = 443;
23   public ServerSocket getServer() throws Exception {
24         KeyStore ks1 = KeyStore.getInstance("JKS");
25   ks1.load(new FileInputStream(key_store), key_storepass);
26         KeyManagerFactory keyMgrFact = KeyManagerFactory.getInstance("SunX509");
27         keyMgrFact.init(ks1, keyPasswd);
28         SSLContext sslcnxt =
29   SSLContext.getInstance("SSLv3");
```

```
30   sslcnxt.init(keyMgrFact.getKeyManagers(), null, null);
31         ServerSocketFactory srvSocFac =
32           sslcnxt.getServerSocketFactory();
33   SSLServerSocket sslSrvSkt = (SSLServerSocket)
34           srvSocFac.createServerSocket(HTTPS_PORT);
35         return sslSrvSkt; }
36   //multi-threading create a new connection for each request
37      public void run() {
38         ServerSocket hear;
39         try {
40            hear = getServer();
41            while(true) {
42               Socket sClient = hear.accept();
43               ProcessConnection conn = new
44                 ProcessConnection(sClient);
45         } } catch(Exception e) {
46   System.out.println("Exception: "+e.getMessage());
47         } }
48      // main program
49      public static void main(String args[]) throws Exception
50      {
51         HttpsServer httpsrv = new HttpsServer();
52         httpsrv.run();
53      }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Key Management

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Attack Scenario: Poor Key Management

- In the present scenario, most of the attacks are aimed at **key management** rather than **cryptographic algorithms**

- **Poor Key Management Threats include:**
  - Mishandling of **Keys**
  - Incorrect Implementation of **Key Generation**
  - Confidentiality **Compromise**
  - Unauthorized use of **Public** or **Secret Keys**
  - Compromise of **Authenticity**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Keys and Certificates

In Java, there is a separate `java.security.cert.Certificate` **class** that manages all interactions with certificates using the **Certificate** Class, the **CertificateFactory** Class and the **X509Certificate** Class

Keys and certificates are often associated with some person or an organization and they deal in how keys are **stored**, **transmitted** and **shared** in the Java Security package i.e., JCE

Keys are an important component of many Java cryptographic algorithms, generally used for **generating** and **verifying** digital signatures or performing encryption

- Asymmetric keys (public and private keys) are used for **digital signatures**
- Symmetric keys (secret keys) are associated with implementation of **certificates**

The main classes that operate on keys include:

- The `KeyPairGenerator` class (generates keys)
- The `KeyFactory` class (interprets key objects and their external representations)

Certificates are used for **authenticating keys** that are used for digital signing (using secret keys or private keys)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Key Management System

- The key management system provides **private keys** to create digital signatures and **public keys** for verifying the digital signatures
- Key management system manages keys (**public**, **private**, **secret keys**) contained within a certificate, stores those keys and retrieves them programmatically using tools

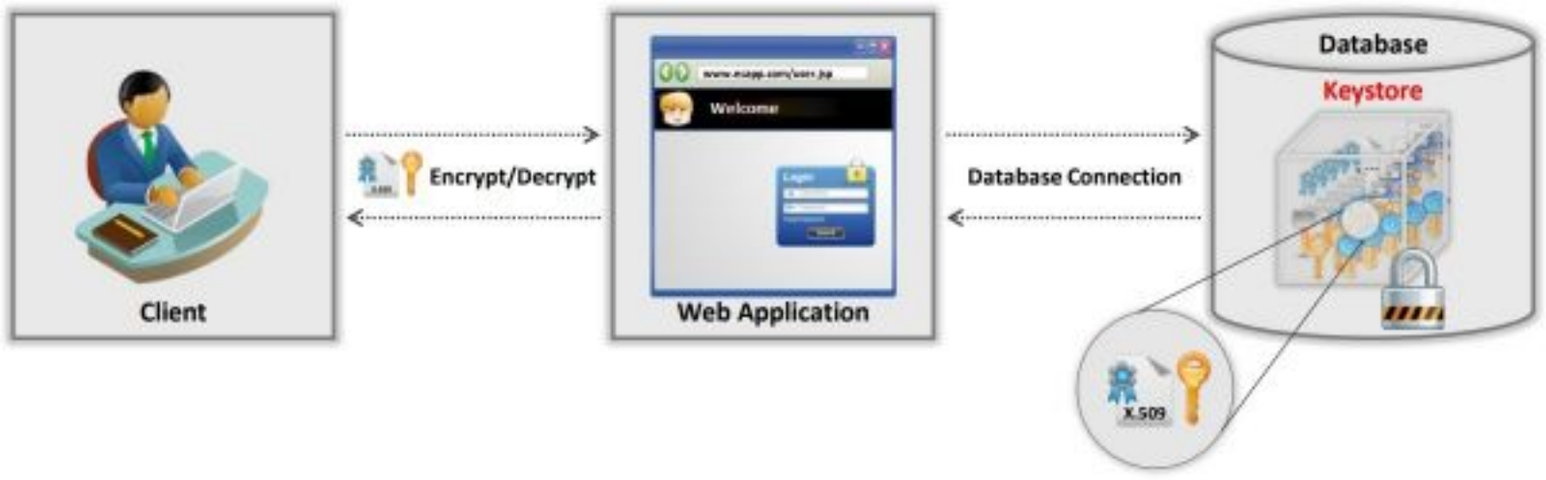
- **Elements of key management system include:**

Keys	Used for **signing data** e.g., JAR files that are usually provided by an entity (an individual or an organization) and include **public key**, **private key** or both public and private keys
Certificates	Used for verifying **digital signatures** (public key) provided by an entity
Identities	Used for **managing identities** with their keys that are stored in the key database (**keystore**)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

KeyStore

- KeyStore provides standard mechanism for managing and storing **cryptographic keys** and **certificates** (password-protected database)
- KeyStore is an important class in **JCA** that manages Java's key management system (`java.security.KeyStore`)
- KeyStore implements entries using `KeyStore.Entry` interface for managing entries such as `KeyStore.PrivateKeyEntry` `KeyStore.SecretKeyEntry` and `KeyStore.TrustedCertificateEntry`



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Implementation Method of KeyStore Class

Methods	Description
`aliases()`	Lists all the alias names of this keystore
`getCertificate(String alias)`	Returns the certificate associated with the given alias
`getCertificateAlias(Certificate cert)`	Returns the (alias) name of the first keystore entry whose certificate matches the given certificate
`getInstance(String type)`	Generates a keystore object of the given type
`getKey(String alias, char[] password)`	Returns the key associated with the given alias, using the given password to recover it
`getType()`	Returns the type of this keystore
`load(InputStream stream, char[] password)`	Loads this KeyStore from the given input stream
`size()`	Retrieves the number of entries in this keystore
`store(KeyStore.LoadStoreParameter param)`	Stores this keystore using the given LoadStoreParameter
`store(OutputStream stream, char[] password)`	Stores this keystore to the given output stream, and protects its integrity with the given password

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

KeyStore: Persistent Data Stores

- Applications that store cryptographic keys in **persistent** and **temporary data stores** or memory are prone to attacks from unauthorized users
- If an application does not securely guard the data store location, an attacker can use insecure **keystore bugs** to read data related to cryptographic keys and certificates
- To protect **keystore** from this type of attack, developers should securely **manage data** in both temporary and persistent data stores
- In the below insecure code, **keystore files** and **registry keys** are accessed due to insecurely managed permissions

MyKeystore.java

```
1  package com.domo;
2
3  import java.io.File;
4  import java.io.FileInputStream;
5  import java.io.IOException;
6  import java.security.KeyStore;
7  import java.security.KeyStoreException;
8  import java.security.NoSuchAlgorithmException;
9  import java.security.cert.CertificateException;
10 public class MyKeystore {
11     public static void main(String[] args) throws KeyStoreException, NoSuchAlgorithmException, CertificateException,
12         KeyStore key=KeyStore.getInstance("JKS");
13         String file = System.getProperty("java.home") +"/lib/security/myKeyStore.jks";
14         FileInputStream fstream = new FileInputStream(new File(file));
15                 key.load(fstream, "storeit".toCharArray());
16     }
17 }
```

- Secure code solution for the above code is to give restricted or no permission to the folder `java.home/lib/security` that holds the storage facility for all keystore files and registry files

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Key Management Tool: KeyTool

- The `java.security.KeyStore` provides a repository for cryptographic keys and certificates
- **Keytool** is an **inbuilt tool** that is used to generate **key pairs, import digital certificates, export existing keys, create self-signed certificates**, etc.

Commands for Generating

- **Generate a Java keystore and key pair**
  - `keytool -genkey -alias mydomain -keyalg RSA -keystore keystore.jks`
- **Generate a certificate signing request (CSR) for an existing Java keystore**
  - `keytool -certreq -alias mydomain -keystore keystore.jks -file mydomain.csr`
- **Generate a keystore and self-signed certificate**
  - `keytool -genkey -keyalg RSA -alias selfsigned -keystore keystore.jks -storepass password -validity 360`

Commands for Importing

- **Import a root or intermediate CA certificate to an existing Java keystore**
  - `keytool -import -trustcacerts -alias root -file Thawte.crt -keystore keystore.jks`
- **Import a signed primary certificate to an existing Java keystore**
  - `keytool -import -trustcacerts -alias mydomain -file mydomain.crt -keystore keystore.jks`

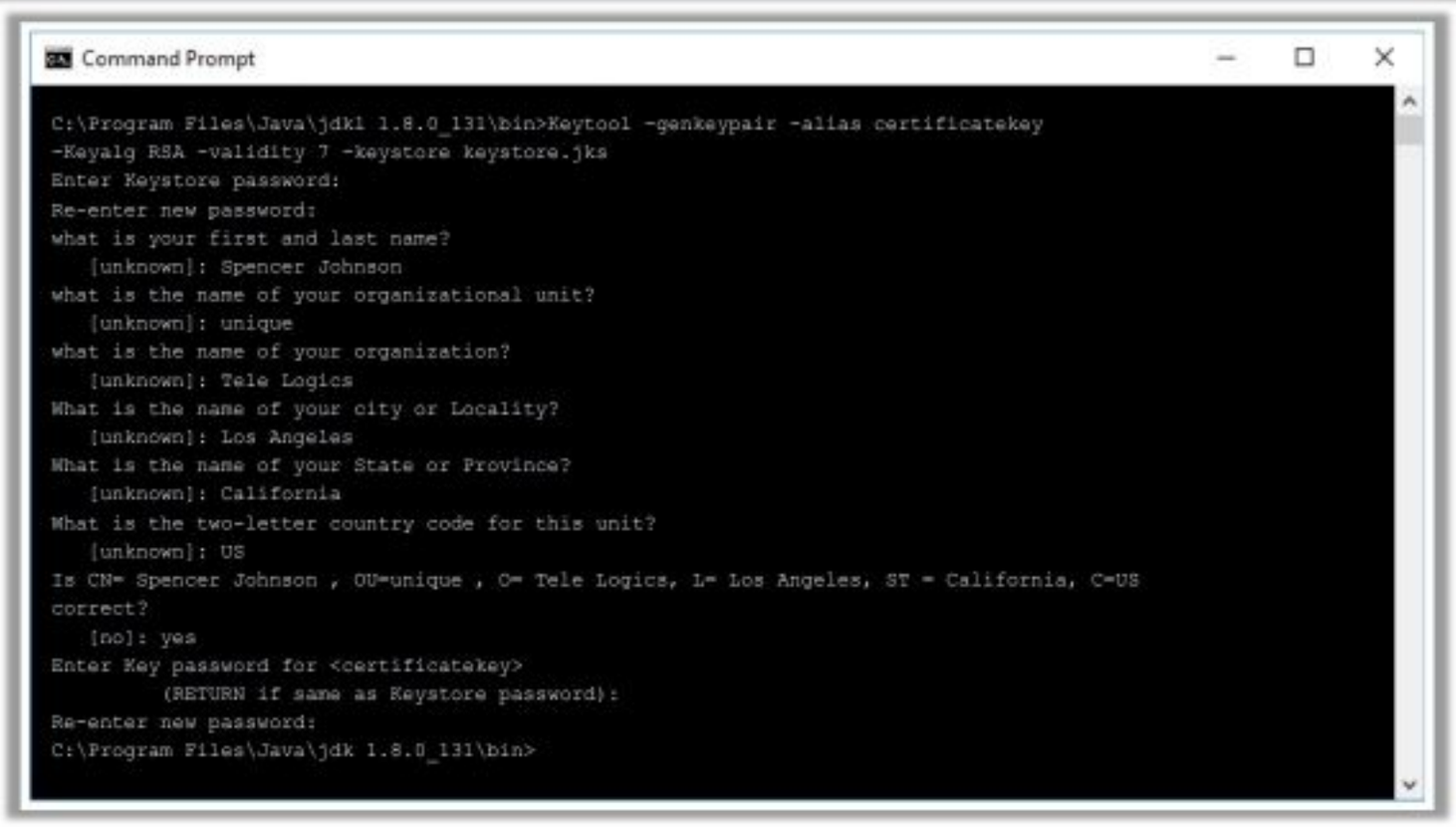
Commands for Checking

- **Check a stand-alone certificate**
  - `keytool -printcert -v -file mydomain.crt`
- **Check which certificates are in a Java keystore**
  - `keytool -list -v -keystore keystore.jks`
- **Check a particular keystore entry using an alias**
  - `keytool -list -v -keystore keystore.jks -alias mydomain`

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Key Management Tool: KeyTool (Cont'd)

- Generates a **private key** in the keystore:
  - **`keytool -genkeypair -alias certificatekey -keyalg RSA -validity 7 -keystore keystore.jks`**



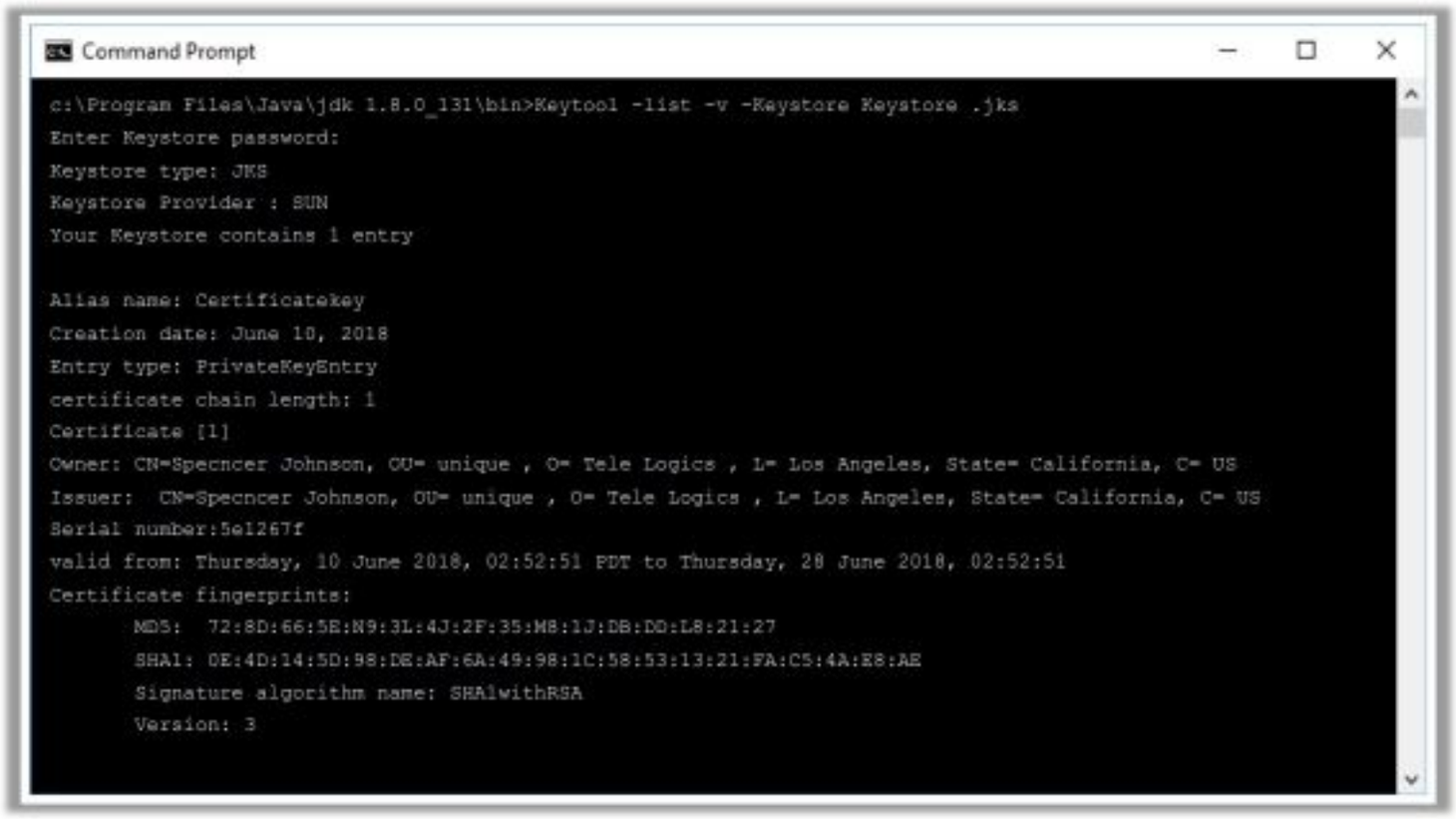
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Key Management Tool: KeyTool (Cont'd)

- Displays **contents** of the keystore:
  - `keytool -list -v -keystore keystore.jks`



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Digital Certificates

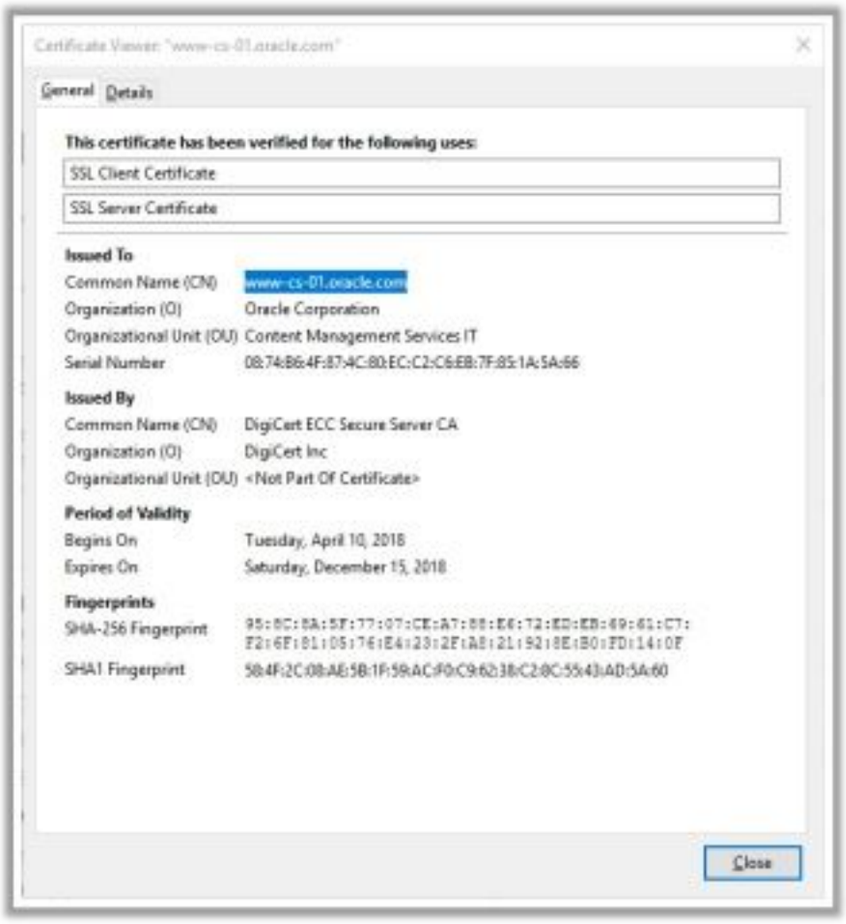
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Digital Certificates

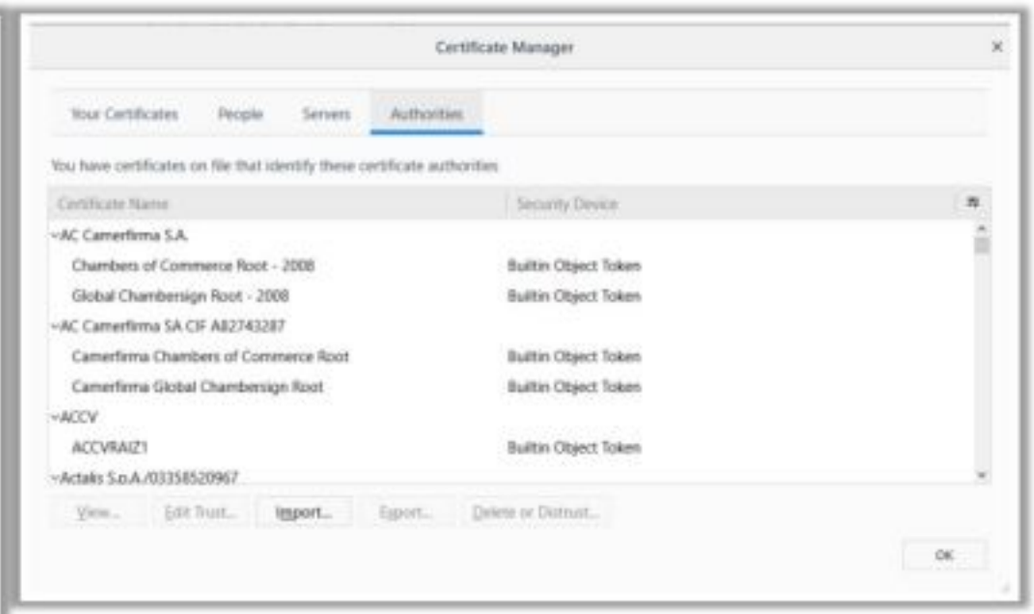
- Digital certificates are used for **identifying** the **author** (who created the keys) and generally issued by a certification authority (CA)
- A Digital certificate includes:
  - User (entity) Information
  - User's public key
  - Digital signature of the CA
  - Issue and expiry date
- Types of Digital Certificates
  - Secure Socket Layer (SSL) Server certificates
  - Code Signing certificates
  - Client Certificates



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certification Authorities

- In Java, there are defined sets of trusted certificates provided by some trusted certification authorities such as **VeriSign**, **Entrust** or **Thawte**
- The Java platform has a special inbuilt key store, cacert that contains certificates from trusted CAs (`$JREHOME/lib/security/cacerts`)
- In Java, the class that represents digital certificates is `java.security.cert.Certificate`
- Example: ANSI standards and X509 certificates use **distinguished name** (DN) convention to identify the entities that include
  - commonName (CN)
  - organizationUnit (OU)
  - organizationName (O)
  - localityName (L)
  - stateName (S)
  - country (C)



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Certification Authorities (Cont'd)



Source: *http://www.comodo.com*



Source: *http://www.thawte.com*



Source: *http://www.verisign.com*

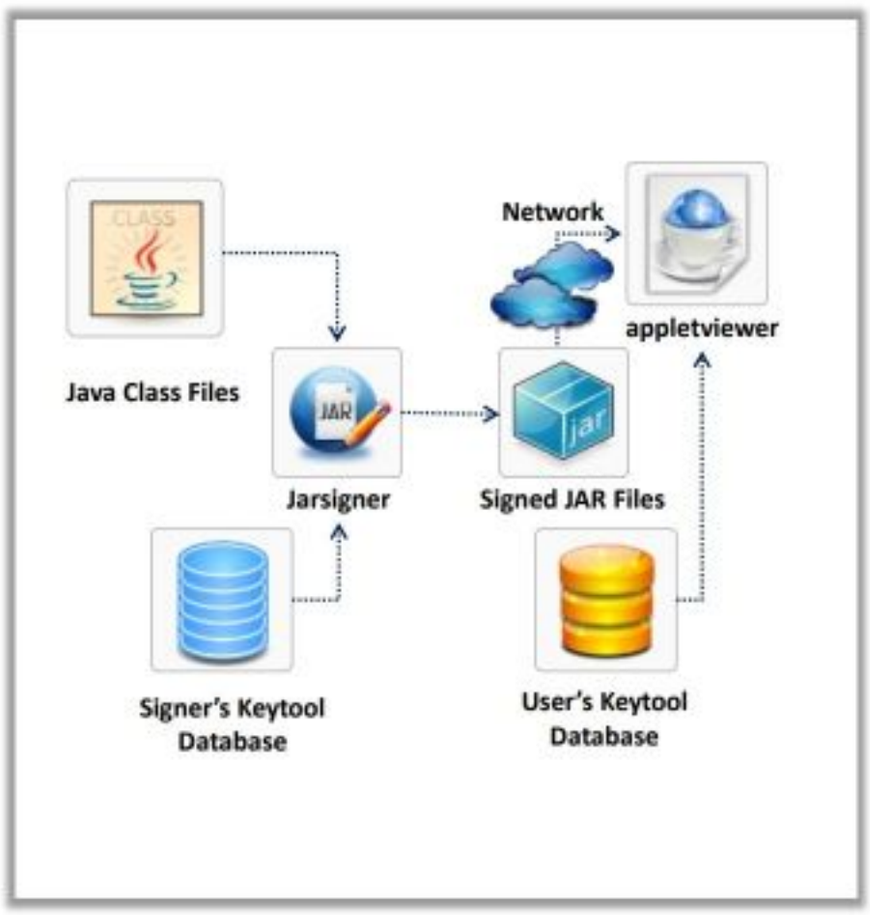


Source: *http://www.entrust.net*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Signing Jars

- Java Archive (JAR) is a file format that contains multiple files i.e., **class files** and **subsidiary resources** associated with applets and applications
- It includes two important concepts:
  - **Signing JAR files**
    - Signing Jars is an effective way of conveying **trustworthiness** of a **JAR** through digital signature ensuring that the public key is using a certificate from a trusted CA
    - To sign a JAR file, a **private key** and its associated **public key** are required
  - **Verifying signed JAR files**
    - Verifying signed JAR implies that the signed JAR does not contain **malicious data** and also ensures that it is not modified or manipulated by untrusted sources



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Signing JAR Tool: Jarsigner

- Jarsigner is an inbuilt **command line Java tool** used to sign a JAR file, and it generates a key from **keystore** and verifies the integrity of a **signed JAR file**

To Sign a JAR File

```
jarsigner -keystore keystore-name -
storepass keystore-password -keypass
key-password jar-file alias-name
```

To Verify a Signed JAR File

```
jarsigner -verify jar-file
```

Options	Description
`-keystore url`	Specifies a desired keystore
`- storepass password`	Specifies a keystore password
`-keypass password`	Specifies an alias password
`-sigfile file`	Specifies a base name for .SF and .DSA files (.keystore created files)
`-signedjar file`	Specifies the name of the signed JAR file

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Signing JAR Tool: Jarsigner (Cont'd)

- Signing the application with the **code signing certificate**

Command Prompt

```
s = signature was verified
m = entry is listed in manifest
k = at least one certificate was found in keystore
i = at least one certificate was found in identity scope
jar verified .
C:\Program Files <x86>\Java\jdk1.8.0_131\bin>keytool -list -v -storetype pkcs12 -
keystore "c:\validation\AppSigning\MSI_Certificate\ComodoCertificate.pfx"
Enter keystore password:
keystore type: PKCS12
keystore provider: SunJSSE
Your keystore contains 1 entry
Alias name: 1e-4a44ed1-e260-4076-84d4-39b96708b885
Creation date: June 10, 2018
Entry type: PrivatekeyEntry
Certificate chain length: 2
Certificate [1]:
Owner: CN = Digital Techno Solutions, O= Digital Techno Solutions, Street =53 Mansion House,
       L=Dallas, ST=Texas, OLD.2.5.4.17=95054-1537, C =US
Issuer : CN= Digi Era Technologies, OU= http:// www.Digieratechnologies.com,O = Digi Era Technologies,
       L=Chicago, ST=Illinois, C=US
Serial number:1f393032a6ca5e03766d3829f970adf2
Valid from: Mon Jun 10 17:00:00 PDT 2018 until: wed Jun 10 16:59:59 PDT 2019
Certificate fingerprints:
          MD5:81:4A:EE:74:45:5E:76:D8:80:75:1D:A4:97:01:3A:49
          SHA1:4F:53:42:27:c6:E7:06:8B:EB:52:D5:2B:D4:2E:3A:07:1D:47:17:7F
          Signature algorithm name : SHA1withRSA
          version: 3
Extensions:
#1: objectId: 2.5.29.15 Critically =true
KeyUsage [
      DigitalSignature
]
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Signing JAR Tool: Jarsigner (Cont'd)

- Verifying the **signed application**

Command Prompt

```
s = signature was verified
c:\Program Files\Java\jdk1.8.0_131\bin>jarsigner -verify -verbose -certs c:\Users
\Peter\Desktop\Articles\Article\CalculatorBad\jar
          488 Fri May 27 11:23:40 PDT 2018 META-INF/MANIFEST .MF
          433 Fri May 27 11:23:40 PDT 2018 META-INF/LE-4A446.SF
         3000 Fri May 27 11:23:40 PDT 2018 META-INF/LE-4A446.RSA
sm     128235 Tue May 03 15:01:08 PDT 2018 CalculatorBad.jar

       X.509 , CN = Digital Techno Solutions, 0= Digital Techno Solutions, STREET =53 Mansion House,
       L=Dallas, ST=Texas, OLD.2.5.4.17=95054-1537, C =US,
       [certificate is valid from 6/22/18 5:00 PM to 6/22/19 4:59 PM]
       X.509, CN= Digi Era Technologies, OU= http:// www.Digieratechnologies.com,0 = Digi Era Technologies,
       L=Chicago, ST=Illinois, C=US
        [certificate is valid from 6/9/18 11:31 AM to 7/9/19 11:40 AM]

sm     118965 Tue May 03 15:01:08 PDT 2011 CalculatorBad.jar_lib/appup_sdk_java_v1_.1. jar

       X.509 , CN = Digital Techno Solutions, 0= Digital Techno Solutions, STREET =53 Mansion House,
       L=Dallas, ST=Texas, OLD.2.5.4.17=95054-1537, C =US,
       [certificate is valid from 6/22/18 5:00 PM to 6/22/19 4:59 PM]
       X.509, CN= Digi Era Technologies, OU= http:// www.Digieratechnologies.com,0 = Digi Era Technologies,
       L=Chicago, ST=Illinois, C=US
        [certificate is valid from 6/9/18 11:31 AM to 7/9/19 11:40 AM]

      s = signature was verified
      m = entry is listed in manifest
      k = at least one certificate was found in keystore
      i = at least one certificate was found in identity scope
jar verified
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Signed Code Sources

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Signed Code Sources

- Code signing is a security mechanism that performs digital **signing** of **Java scripts** and **executables** using cryptography algorithms to prevent malicious activities
- The object `java.security.CodeSource` includes a piece of code, `SecureClassLoader` (along with its subclasses) and `CodeSource` class are related with generating, modifying and handling code source objects
- The `CodeSource` class represents an original code execution point that includes (applets) codebase to **encapsulate URL** locations along with certificates used for verification of **signed code**
- Some implementation methods of the `CodeSource` class:

  `public CodeSource(URL url, Certificate cers[])`

  - Creates a codesource for a set of certificates in a specified location
  - Creates a codesource for a set of code signers in a specified code location

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Insecure Code for Signed Code Sources

- The `jarClassLoader` verifies the signature using the public key contained in a JAR file although the authenticity of this signature is still insufficient

JarRunner.java

```
1  package com.domo;
2
3  import java.io.IOException;
4  import java.lang.reflect.InvocationTargetException;
5  import java.lang.reflect.Method;
6  import java.lang.reflect.Modifier;
7  import java.net.JarURLConnection;
8  import java.net.URL;
9  import java.net.URLClassLoader;
10 import java.util.jar.Attributes;
11
12 public class JarRunner {
13 @SuppressWarnings("resource")
14 public static void main(String[] args)
15 throws IOException, ClassNotFoundException,
16 NoSuchMethodException, InvocationTargetException {
17 URL httpUrl = new URL(args[0]);
18 // Create class loader for the application jar file
19 JarClassLoader jcl = new JarClassLoader(httpUrl);
20 // Get the application's main class name
21 String sname = jcl.getMainClassName();
22 // Get arguments for the application
23 String[] argmts = new String[args.length - 1];
24 System.arraycopy(args, 1, argmts, 0, argmts.length);
25 // Invoke application's main class
26 jcl.invokeClass(sname, argmts);
```

```
27 }
28 }
29 final class JarClassLoader extends URLClassLoader {
30 private URL httpUrl;
31 public JarClassLoader(URL httpUrl) {
32 super(new URL[] { httpUrl });
33 this.httpUrl = httpUrl;
34 }
35 public String getMainClassName() throws IOException {
36     URL url = new URL("jar", "", httpUrl + "!/");
37     JarURLConnection juconn = (JarURLConnection) url.openConnection();
38     Attributes attrb = juconn.getMainAttributes();
39     return attrb != null ?
40     attrb.getValue(Attributes.Name.MAIN_CLASS) : null;
41     }
42     public void invokeClass(String sname, String[] args)
43     throws ClassNotFoundException, NoSuchMethodException,
44     InvocationTargetException {
45     Class<?> c1 = loadClass(sname);
46     Method m1 = c1.getMethod("main", new Class[] { args.getClass() });
47     m1.setAccessible(true);
48     int modifs = m1.getModifiers();
49 if (m1.getReturnType() != void.class || !Modifier.isStatic(modifs)
50 ||!Modifier.isPublic(modifs)) {
51     throw new NoSuchMethodException("main");
52 }
53 try {
54 m1.invoke(null, new Object[] { args });
55 } catch (IllegalAccessException e) {
56 System.out.println("Access denied");
57     } } }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secure Code for Signed Code Sources

1. Secure code using Jarsigner

- In the code, `-verify` option (jarsigner) is used explicitly to check the JAR file signature at the command line
- `jarsigner -verify signed-updates-jar-file.jar`

2. Secure code using Certificate Chain

- Invoke class method is used to verify the signature i.e., by obtaining a chain of certificates from **CodeSource** class

JarRunner.java

```
67  public void invokeClass(String sname, String[] args)
68  throws ClassNotFoundException, NoSuchMethodException,
69  InvocationTargetException, GeneralSecurityException,
70  IOException {
71  Class<?> c1 = loadClass(sname);
72  Certificate[] certf =
73  c1.getProtectionDomain().getCodeSource().getCertificates();
74  if (certf == null) {
75  // return, do not execute if unsigned
76  System.out.println("No signature!");
77  return;
78  }
79  KeyStore kyst = KeyStore.getInstance("JKS");
80  kyst.load(new FileInputStream(System.getProperty("user.home"+ File.sepa
81      "loadkeystorepassword".toCharArray());
82      // user is the alias
83      Certificate pubCertf = kyst.getCertificate("user");// check with th
84      certf[0].verify(pubCertf.getPublicKey());
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Hashing

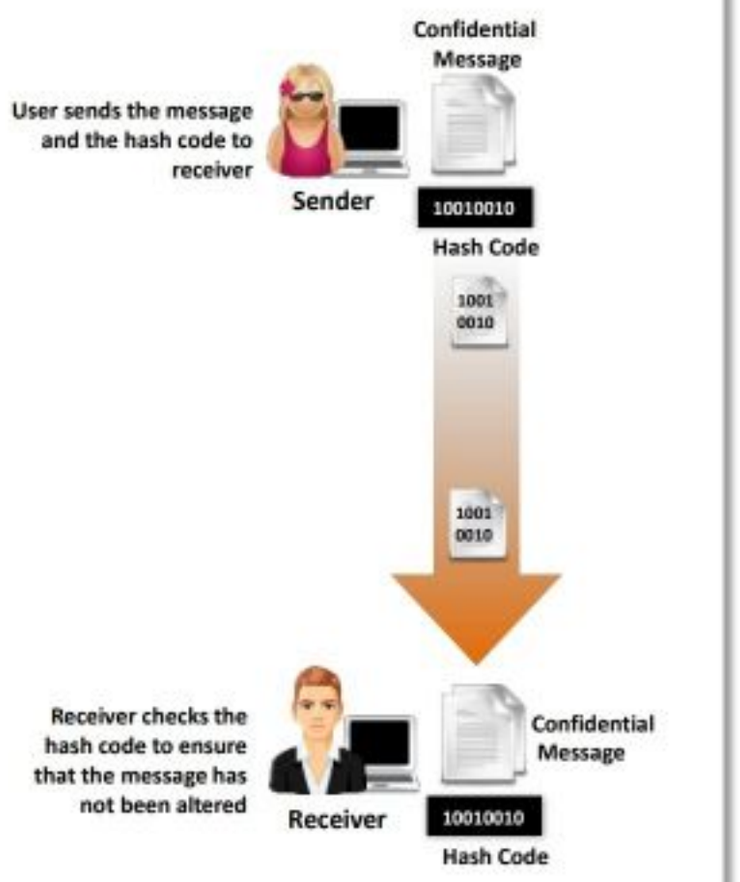
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Hashing

- Hashing is one of the forms of **cryptography** that transforms the information into a **fixed-length value** or key that represents the original information
- The hashing technique ensures the **security of information** by checking the **integrity of information** on both the sender and receiver sides
- Checking the integrity of information:
  - The sender of the message creates a **hash code** of it and sends the message to the **receiver** along with its **hash code**
  - The receiver again creates a **hash code** for the same messages at the **receiver side** and compares both the hash codes; if it is a match, then the message is not tampered in the **transmission**. If they do not match, then message is tampered with by third parties during transmission



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Hashing Algorithms

- To implement hashing in Java include **java.security.MessageDigest** class
- **MD2, MD5, SHA-1, SHA-265, SHA-384, SHA-512, PBKDF2, bcrypt**, and **scrypt** are some of the hashing algorithms used

Example: Hashing using MD5

Hashing.java

```
1  package com.domo;
2
3  import java.security.MessageDigest;
4
5  public class Hashing {
6  public static void main(String[] args) {
7
8      String passwordToHash="Passwrord@123";
9      String GeneratedPassword=null;
10     try {
11         MessageDigest md=MessageDigest.getInstance("MD5");
12         md.update(passwordToHash.getBytes());
13         byte[] bytes=md.digest();
14         StringBuilder sb=new StringBuilder();
15         for(int i=0;i<bytes.length;i++)
16         {
17             sb.append(Integer.toString((bytes[i]&0xff)+0x100,16).substring(1));
18         }
19         GeneratedPassword=sb.toString();
20         System.out.println("Hashed password="+GeneratedPassword);
21
22     } catch (Exception e) {
23         e.getMessage();
24     }}}
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Securing Hashed Password with Salt

- Hashing the same password results in the same hash every time which makes it **vulnerable** to **Dictionary** and **Brute Force Attacks**
- Rainbow tables, lookup tables and reverse lookup tables are used to **crack Hash codes**
- By adding a random string (**salt**) before implementing hashing results in generating random hash for the same string (**password**)

Do's and Don'ts while Implementing Salting

- Do not use **hard-coded salt**
- Re-Generate a **unique salt** each time
- Ensure that the size of the salt is big enough to generate numerous possible salts, so that the attacker cannot generate a Lookup table to **crack the passwords**
- Do not implement **double hashing**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Securing Hashed Password with Salt (Cont'd)

Example: Implementing Salting before Hashing

Hashing.java

```
1  package com.domo;
2
3  import java.security.MessageDigest;
4  import java.security.NoSuchAlgorithmException;
5  import java.security.SecureRandom;
6
7  public class Hashing {
8  public static void main(String[] args) throws NoSuchAlgorithmException {
9
10     String passwordToHash="Password@123";
11     String GeneratedPassword=null;
12     byte[] salt = getSalt();
13     try {
14         MessageDigest md=MessageDigest.getInstance("MD5");
15         md.update(salt);
16         byte[] bytes=md.digest();
17         StringBuilder sb=new StringBuilder();
18         for(int i=0;i<bytes.length;i++)
19         {
20             sb.append(Integer.toString((bytes[i]&0xff)+0x100,16).substring(1));
21         }
22         GeneratedPassword=sb.toString();
23         System.out.println("Hashed password="+GeneratedPassword);
24
25     } catch (Exception e) {
26         e.getMessage();
27     }}
28
29 private static byte[] getSalt() throws NoSuchAlgorithmException {
30     SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
31     byte[] salt = new byte[16]; sr.nextBytes(salt);     return salt;
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Implementing Hashing with Salt in Spring Security

- Spring uses the following interfaces for **password hashing** and **salting**
  - `org.springframework.security.providers.encoding.PasswordEncoder`
  - `org.springframework.security.authentication.dao.SaltSource`

Vulnerable Bean Configuration

spring-security.xml

```
1 <?xml version = "1.0" encoding = "UTF-8"?>
2
3 <beans xmlns = "http://www.springframework.org/schema/beans"
4    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
5    xsi:schemaLocation = "http://www.springframework.org/schema/beans
6    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7
8
9  <bean id="passwordEncoder" class="org.springframework.security.authentication.encoding.ShaPasswordEncoder">
10  <constructor-arg value="160"/>
11 </bean>
12 <bean id="saltSource" class="org.springframework.security.authentication.dao.SystemWideSaltSource">
13  <property name="systemWideSalt" value="#Saltvalue;" />
14 </bean>
15 </beans>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Implementing Hashing with Salt in Spring Security(Cont'd)

- Define a bean configuration in **applicationContext.xml** file to implement hashing
- **ShaPasswordEncoder** by default generates a 160 bit hash. Change the **constructor-arg** value to 256 to generate a 256 bit hash
- **systemWideSalt** property defines the value to be used as salt. Include a long salt for generating random salt

Secure Bean Configuration

spring-security.xml

```
1 <?xml version = "1.0" encoding = "UTF-8"?>
2 <beans xmlns = "http://www.springframework.org/schema/beans"
3    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
4    xsi:schemaLocation = "http://www.springframework.org/schema/beans
5    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
6 <bean id="passwordEncoder" class="org.springframework.security.authentication.encoding.ShaPasswordEncoder">
7 <constructor-arg value="256"/>
8 </bean>
9 <bean id="saltSource" class="org.springframework.security.authentication.dao.SystemWideSaltSource">
10  <property name="systemWideSalt" value="#This,Is9087%49A3Salt*Value;" />
11  </bean>
12   </beans>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

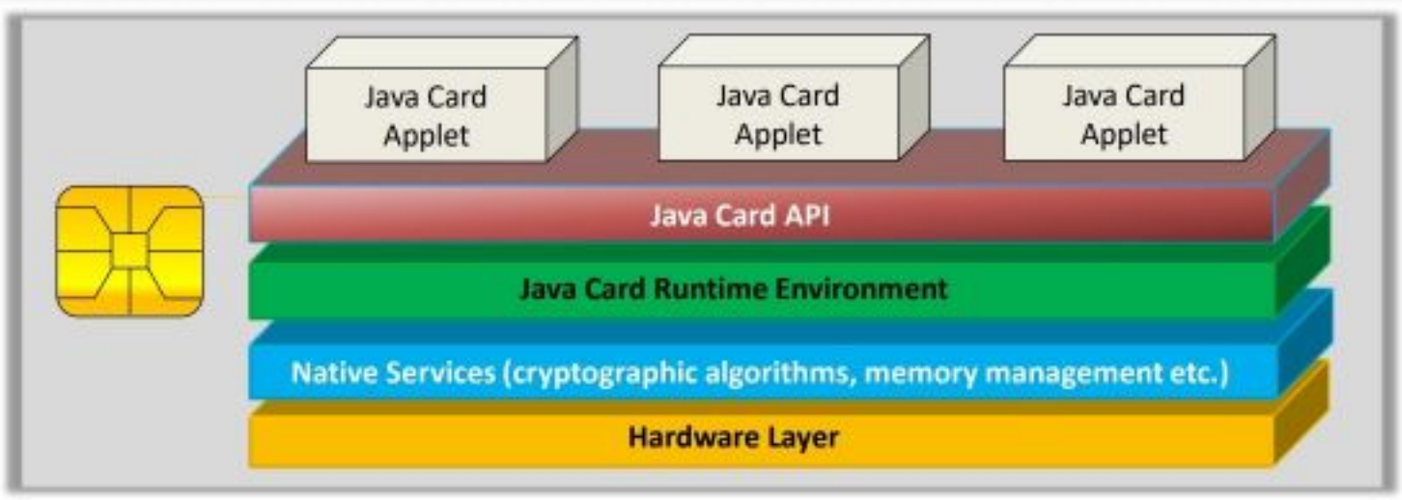
@ErrorCatchers

Java Card Cryptography

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java Card Cryptography

- Java Card specifications includes Java Card Virtual Machine specification(**JCVM**), Java Card Runtime Environment Specification (**JCRE**) and Java Card Application Programming Interface (**API**)
- Java Card Applet is a kind of Java application running on similar resource-constrained devices with a subset specifications of Java technology
- There are three major APIs defined in Java card platform:
  - javacard.framework
  - javacard.security
  - javacardx.crypto



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Java Card Cryptography (Cont'd)

javacard.security

Interfaces

- AESKey
- DESKey
- DHKey
- DHPrivateKey
- DHPublicKey
- DSAKey
- DSAPrivateKey
- DSAPublicKey
- ECKey
- ECPrivateKey
- ECPublicKey
- HMACKey
- Key
- KoreanSEEDKey
- PrivateKey
- PublicKey
- RSAPrivateCrtKey
- RSAPrivateKey
- RSAPublicKey
- SecretKey
- SignatureMessageRecovery

Classes

- Checksum
- InitdializedMessageDigest
- InitdializedMessageDigest.OneShot
- KeyAgreement
- KeyBuilder
- KeyPair
- MessageDigest
- MessageDigest.OneShot
- RandomData
- RandomData.OneShot
- Signature
- Signature.OneShot

javacardx.crypto

Interfaces

- KeyEncryption

Classes

- AEADCipher
- Cipher
- Cipher.OneShot

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java Card Cryptography (Cont'd)

Generate Random Number in Applet on Smart Card

PasswordEncoding.java

```
27      {
28          //Generate Random Number
29          byte[] random = JCSystem.makeTransientByteArray((short)8, JCSystem.CLEAR_ON_DESELECT);
30          RandomData generator = RandomData.getInstance((byte)RandomData.ALG_FAST);
31          generator.setSeed(random, (short)0, (short)8);
32          generator.nextBytes(random, (short)0, (short)8);
```

Generate Message Digest in Applet on Smart Card

Quick Access

PasswordEncoding.java

```
11      public void numgen()
12      {
13      //Generate Random Number
14      byte[] digest = JCSystem.makeTransientByteArray((short)20, JCSystem.CLEA
15      MessageDigest alg = MessageDigest.getInstance(MessageDigest.ALG_SHA, fal
16      //buffer is the tmp data property (byte[]) in applet
17
18      alg.doFinal(buffer, (short)5, (short)8, digest, (short)0);
19      }
20
21  }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Java Card Cryptography (Cont'd)

Generate/Verify RSA signature in Applet on Smart Card

```
41    public boolean verifyRSAsign()
42    {
43        //Create KeyPair Object
44        KeyPair kp = new KeyPair(KeyPair.ALG_RSA, (short)512);
45        //Generate KeyPair
46        kp.genKeyPair();
47        RSAPrivateKey privateKey = (RSAPrivateKey)kp.getPrivate();
48        RSAPublicKey publicKey = (RSAPublicKey)kp.getPublic();
49        //sign
50        Signature sig = Signature.getInstance(Signature.ALG_RSA_SHA_PKCS1, false);
51        sig.init(privateKey, Signature.MODE_SIGN);
52        sig.sign(dataBytes, (short)0, (short)dataBytes.length, signedBytes, (short)0);
53        //verify signature
54        sig.init(publicKey, Signature.MODE_VERIFY);
55    boolean verify = sig.verify(signedBytes, (short)0, (short)signedBytes.length, buffer, (short)0, (short)(publicKey.getSize()/8));
56        return verify;
57
58    }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java Card Cryptography (Cont'd)

Generate / Verify ECC signature in Applet on Smart Card

PasswordEncoding.java

```
70
71    //Create KeyPair Object
72    KeyPair kp = new KeyPair(KeyPair.ALG_EC_FP, KeyBuilder.LENGTH_EC_FP_192);
73    //Generate KeyPair
74    kp.genKeyPair();
75    ECPrivateKey privateKey = (ECPrivateKey)kp.getPrivate();
76    ECPublicKey publicKey = (ECPublicKey)kp.getPublic();
77    //sign
78    Signature sig = Signature.getInstance(Signature.ALG_ECDSA_SHA, false);
79    sig.init(privateKey, Signature.MODE_SIGN);
80    //buffer is the OUT data container (byte[])
81    sig.sign(dataBytes, (short)0, (short)dataBytes.length, signedBytes, (short)0);
82    //verify signature
83    sig.init(publicKey, Signature.MODE_VERIFY);
84    boolean verify = sig.verify(signedBytes, (short)0, (short)signedBytes.length, buffer, (short)0, (short)(publicKey.getSize()/8));
```

.AES in Applet on Smart Card

PasswordEncoding.java

```
95    public void AESinApplet()
96    {
97        //Create AESKey Object
98        AESKey key = (AESKey) KeyBuilder.buildKey(KeyBuilder.TYPE_AES, KeyBuilder.LENGTH_AES_128, false);
99        //Create Cipher object
100       Cipher cipher = Cipher.getInstance(Cipher.ALG_AES_BLOCK_128_CBC_NOPAD, false);
101       //encryption
102       cipher.init(key, Cipher.MODE_ENCRYPT);
103       cipher.doFinal(dataBytes, (short)dataOffset, (short)dataLength, encryptedBytes, (short)0);
104       //decryption
105       cipher.init(key, Cipher.MODE_DECRYPT);
106       cipher.doFinal(encryptedBytes, (short)0, (short)encryptedLength, dataBytes, (short)0);
107
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Spring Security: Crypto Module

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Crypto Module

- Spring **crypto module** provides classes for
  - symmetric encryption
  - key generation
  - password encoding

Encryptors

Set of classes for creating

- **ByteEncryptor**
  - Uses **256-bit AES using PKCS #5's PBKDF2 (Password-Based Key Derivation Function #2)** for encryption
  - Returns encrypted data in **raw byte[]** form
- **TextEncryptor**
  - Implements standard **BytesEncryptor** for encryption and returns result as hex-encoded strings
- **Queryable TextEncryptor**
  - The initialization vector for encryption is constant and generates the **same encryption result** for the same text
  - It should be implemented when the previously generated hash needs to be **matched** with the generated hash for the **same text**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Spring Security Crypto Module: Key Generators

- Set of classes for creating following types of key generators
  - **BytesKeyGenerator**
    - generates **byte[]** keys
    - **KeyGenerators.secureRandom()** method generates a key of **8 bytes** by default
    - Specify keysize while implementing **KeyGenerators.secureRandom(16)** for generating large size keys
    - **KeyGenerators.shared()** method generates the **same key** every time
  - **StringKeyGenerator**
    - generates **hex-encoded string** keys
    - **KeyGenerators.string()** generates **8-byte** long string key by default

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Spring Security Crypto Module: PasswordEncoder

- Provide password encoding feature by implementing **PasswordEncoder** Interface
- **StandardPasswordEncoder**, **Md5PasswordEncoder** and **BCryptPasswordEncoder** are some of the password encoders supported in spring security
  - **StandardPasswordEncoder**
    - Combines a plain password with a site-wide secret and 8-byte random salt, and implements SHA-256 hashing algorithm with 1024 iterations
    - **Site-wide Key** should not be stored along with the passwords to prevent attackers from gaining access to the stored passwords using a brute force attack
    - 1024 iterations enables creating a unique and strong key
    - The **random salt** ensures that a unique hash is generated ~~on~~ when the same text is supplied multiple times
  - **BCryptPasswordEncoder**
    - **BCryptPasswordEncoder** implements bcrypt hashing function based on Blowfish cipher
    - **Bcrypt** generates random salt to generate unique encoded hash of length 60 characters for the same text

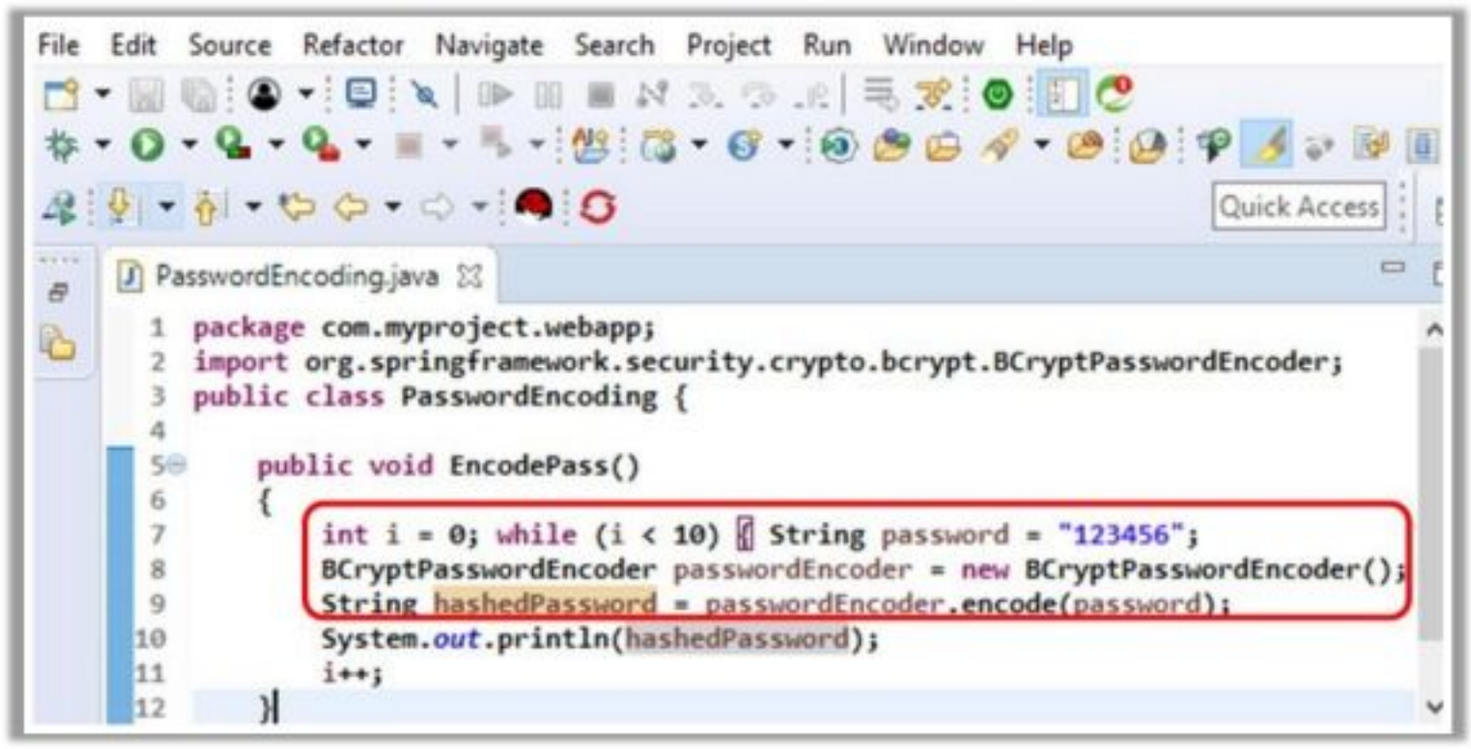
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Implementing BCryptPasswordEncoder()

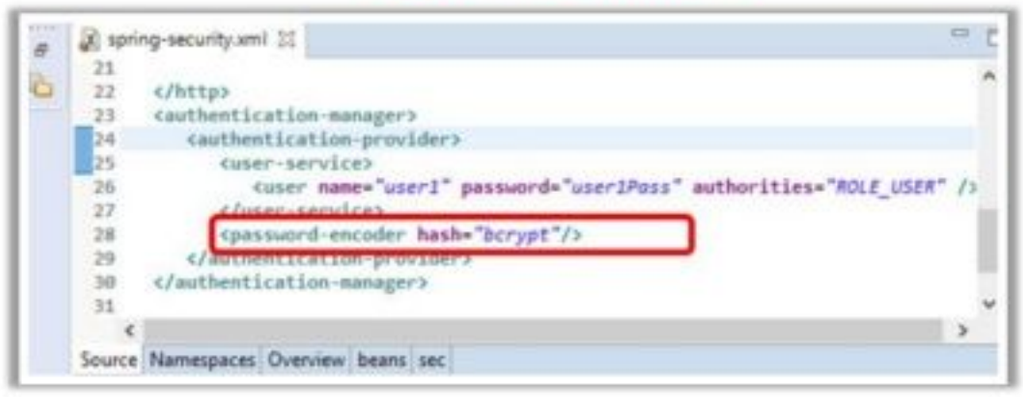
- Sample Code for Implementing **BCryptPasswordEncoder()**



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

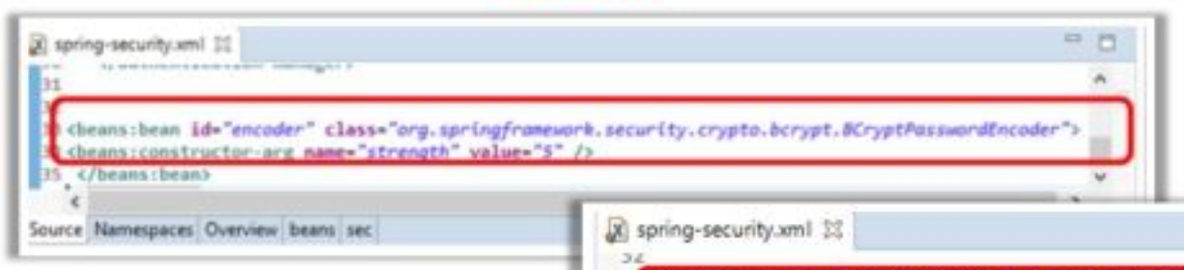
Configuring BCryptPasswordEncoder() in Spring Security

- Configure the authentication-provider to implement **BCryptPasswordEncoder** hashing function in **spring.security.xml**



- To secure the **BCryptPasswordEncoder** give a high value to the strength parameter of the BCryptPasswordEncoder

Vulnerable



Secure



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

JavaScript Object Signing and Encryption (JOSE)

JOSE is a framework for implementing **Cryptography** to **JSON messages**

Used for **Client authentication** and **authorization**

Supported by various programming languages

- **JWS: JSON Web Signature**
  - Securing JSON data using digital signature or Message Authentication Code (MAC) using base64 URL encoding
  - Used for Authorization headers and URI query parameters
  - Simple and compact
  - No canonicalization

  JWS Format: <Header> + <.> + Payload + <.> + Signature.
- **JWE: JSON Web Encryption**
  - Encryption of JSON
  - Complicated

  JWE Format: <Header> + <.> + Encrypted Key + <.> + Initialization Vector + <.> + Cipertext + <.> + Authentication Tag
- **JWT: Java Web Token**
  - JWT Is a JWS or / and JWE with JSON claim (Name / value pair) transferred between clients
- **JWK: JSON Web Key**
  - Public cryptographic keys containing metadata
  - Can be added to JWS, JWE or JWT header
  - Published at HTTPS endpoint
  - Replacement of self signed certificate
  - Saved in files

- To implement JOSE in Spring include the following packages
  - **org.springframework.security.oauth2.jwt**
  - **org.springframework.security.oauth2.jose**
- **JWA : JSON Web Algorithms**
  - List of Crypto algorithms used for JWE / JWS
  - The "alg" parameter of JWS denotes the algorithm
  - The "alg" and "enc" parameter of JWE denotes the algorithm

List of Supported Algorithm - JWS

Alg parameter	Algorithm
HS256	HMAC implements SHA-256
H5384	HMAC implements SHA-384
H5512	HMAC using SHA-512
RS256	RSASSA-PKCS1-v1_5 using SHA-256
RS384	RSASSA-PKCS1-v1_5 using SHA-384
RS512	RSASSA-PKCS1-v1_5 using SHA-512
ES256	ECDSA using P-256 and SHA-256
ES384	ECDSA using P-384 and SHA-384
ES512	ECDSA using P-521 and SHA-512
PS256	RSASSA-PSS using SHA-256 and MGF1 with SHA-256
PS384	RSASSA-PSS using SHA-384 and MGF1 with SHA-384
P5512	RSASSA-PSS using SHA-512 and MGF1 with SHA-512
none	No digital signature or MAC value

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Attacks against JWT, JWS and JWE

- **Signature Exclusion Attack:** The attacker removes the signature of a **signed message**, makes it a **unsigned message** and hence, the message is considered as valid
- **Countermeasures:** Implementing no algorithm specifies that the **JWS payload** is insecure or the payload is not secured by **digital signature** or **MAC value**

- **Key Confusion (Algorithm Substitution):** The attacker replaces the key with a **known crypto key** for implementing a **known algorithm**
- **Countermeasures:** Using an **algorithm parameter** in the signature verification function. This will determine the actually implemented algorithm and detect any change found

  Verify (**string token** , **string algorithm** , **string verificationKey**)

- **Bleichenbacher Million Message Attack:** Ciphertext attack on protocols based on the **RSA encryption**, where the attacker sends multiple ciphertexts to the server and studies the received response error messages, timing differences etc. to identify **valid** and **invalid padded messages** and extracts sensitive information in the **payload**
- **Countermeasures:** Implement alternative Optimal Asymmetric Encryption Padding (**OAEP**) technique. Respond consistently to all errors by checking the length of **CEK** and **parity bits.** Random Filling where the **misformatted messages** are considered properly PKCS-1 formatted

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Implementing JWS using Jose4J

- JOSE4J is an **Open source library** provided by **Apache** to implement JWS, JWT, JWK

Generating JWS using Jose4J

JoseImpl.java

```
1  package jaaSimpl;
2
3  import org.jose4j.jwk.EcJwkGenerator;
4  import org.jose4j.jwk.PublicJsonWebKey;
5  import org.jose4j.jws.AlgorithmIdentifiers;
6  import org.jose4j.jws.JsonWebSignature;
7  import org.jose4j.keys.EllipticCurves;
8  import org.jose4j.lang.JoseException;
9
10 public class JoseImpl {
11
12     public void Jose() throws JoseException {
13     PublicJsonWebKey jwk = EcJwkGenerator.generateJwk(EllipticCurves.P256);
14     JsonWebSignature jws = new JsonWebSignature();
15     jws.setAlgorithmHeaderValue(AlgorithmIdentifiers.ECDSA_USING_P256_CURVE_AND_SHA256); jws.setPayload("USA #1!");
16     jws.setKey(jwk.getPrivateKey()); jws.setKeyIdHeaderValue(jwk.getKeyId());
17     String StrDenseSerialization = jws.getCompactSerialization();
18     System.out.println(StrDenseSerialization);
19     }
```

Consuming JWS using Jose4J

```
24  public void Jose1() throws JoseException
25  {
26      JsonWebKey jwk = JsonWebKey.Factory.newJwk("{\"kty\":\"EC\",\" + \"\"kid\":\"my-first-key\",\" "
27              + "+ \"\"x\":\"xlKTWTx76fl9OZou4LHpDc3oHLC_vm-db7mdsFvO1JQ\",\" + "
28              + "\"\"y\":\"3jXBG649Uqf7pf8RHO_jcJ8Jrhy23hjD933i6QEVNkk\",\" +"
29              + " \"\"crv\":\"P-256\"}");
30      String StrDenseSerialization = "eyJhbGciOiJFUzI1NiIsImtpZCI6Im15LWZpcnN0LWtleSJ9." +
31              "VVNBICMxIQ." +
32              "QJGB_sHj-w3yCBunJs2wxKgvZgG2Hq9PA-TDQEbNdTm2Wnj2sUSrBKZJAUREzF1FF25BbrgyohbKdGE1cB-hrA";
33      JsonWebSignature jws = new JsonWebSignature();
34      jws.setCompactSerialization(StrDenseSerialization);
35      jws.setKey(jwk.getKey()); String payload = jws.getPayload(); System.out.println(payload);
36  }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Implementing JWE using Jose4J (Cont'd)

Generating JWE using Jose4J

JoseImpl.java

```
42  public void Jose2() throws JoseException
43  {
44      JsonWebEncryption jwe = new JsonWebEncryption();
45      jwe.setPayload("I actually really like Canada");
46      jwe.setKey(new PbkdfKey("don't-tell-p@ul|pam!"));
47      jwe.setAlgorithmHeaderValue(KeyManagementAlgorithmIdentifiers.PBES2_HS256_A128KW);
48      jwe.setEncryptionMethodHeaderParameter(ContentEncryptionAlgorithmIdentifiers.AES_128_CBC_HMAC_SHA_256);
49      String StrDenseSerialization = jwe.getCompactSerialization();
50      System.out.println(StrDenseSerialization);
51  }
```

Consuming JWE using Jose4J

JoseImpl.java

```
56  public void Jose3() throws JoseException
57  {
58      String StrDenseSerialization = "eyJhbGciOiJQQkVTMi1IUzI1NitBMTI4S1ciLCJlbmMiO"
59              + "iJBMTI4Q0JDLUhTMjU2IiwicDJjIjo4MTkyLCJwMnMiOiJRa2JMUW5pS0xVVFFWUDR sIn0." +
60              "g7s-MxHFn5WHCfO33hgWYiAtH1lB83TnufWoaFIEujEYb14pqeH9Mg." + "6h172lww9VqemjMQMaVPdg."
61              "YMg_F8aoT3ZByou3CURhKzaGX1nc5QJDo3cWyUSyow0." + "Ie4iYLbdQCqwMNJf37rEZg";
62      JsonWebEncryption jwe = new JsonWebEncryption();
63      jwe.setCompactSerialization(StrDenseSerialization);
64      jwe.setKey(new PbkdfKey("don't-tell-p@ul|pam!"));
65      String payload = jwe.getPayload(); System.out.println(payload);
66  }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Implementing JWK using Jose4J (Cont'd)

Generating JWK using Jose4J

JoseImpl.java

```
71    public void Jose04()
72    {
73        List<JsonWebKey> jwkList = new LinkedList<JsonWebKey>();
74        for (int kid = 4; kid < 7; kid++)
75            {
76                JsonWebKey jwk = EcJwkGenerator.generateJwk(EllipticCurves.P256);
77                jwk.setKeyId(String.valueOf(kid)); jwkList.add(jwk);
78            }
79        JsonWebKeySet jwks = new JsonWebKeySet(jwkList);
80        System.out.println(jwks.toJson(JsonWebKey.OutputControlLevel.PUBLIC_ONLY));
81        }
```

Consuming JWK using Jose4J

JoseImpl.java

```
83
84    public void Jose5() throws JoseException
85    {
86        String jwksJson = "{\"keys\":[n\" +"
87                + " \" {\"kty\":\"EC\",n\"kid\":\"4\",n\" + "
88                + "\" \"x\":\"LX-7aQn7RAx3jDDTioNssbODUfED_6XvZP8NsGzMlRo\", n\" +"
89                + " \" \"y\":\"dJbHEoeWzezPYuz6qjKJoRVLks7X8-BJXbewfyoJQ-A\",n\" + "
90                + "\" \"crv\":\"P-256\"},n\" + \" {\"kty\":\"EC\",n\"kid\":\"5\",n\" + "
91                + "\" \"x\":\"f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU\",n\" +"
92                + " \" \"y\":\"x_FEzRu9m36HLN_tue659LNpXN6pCyStikYjKIWI5a0\",n\" +"
93                + " \" \"crv\":\"P-256\"},n\" + \" {\"kty\":\"EC\",n\"kid\":\"6\",n\" +"
94                + " \" \"x\":\"J8z237wci2YJAzAr5dWIj4OgrOCCfuZ18WI77jsiS00\",n\" +"
95                + " \" \"y\":\"5tTxvax8aRMMJ4unKdKsV0wcf3pOI3OG771gOa45wBU\",n\""
96                + " + \" \"crv\":\"P-256\"}n\" + \"]}";
97        JsonWebKeySet jwks = new JsonWebKeySet(jwksJson);
98        JsonWebKey jwk = jwks.findJsonWebKey("5", null, null, null);
99        System.out.println(jwk.getKey());
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Dos and Don'ts in Java Cryptography

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Dos and Don'ts: Avoid using Insecure Cryptographic Algorithms

1. The following code uses cryptographically insecure algorithms such as DES

InsureCryptoAlgo.java

```
1  package com.domo;
2  import javax.crypto.KeyGenerator;
3  import javax.crypto.SecretKey;
4  import javax.crypto.Cipher;
5  import java.security.NoSuchAlgorithmException;
6  import java.security.InvalidKeyException;
7  import java.security.InvalidAlgorithmParameterException;
8  import javax.crypto.NoSuchPaddingException;
9  import javax.crypto.BadPaddingException;
10 import javax.crypto.IllegalBlockSizeException;
11 import sun.misc.BASE64Encoder;
12
13 public class InsureCryptoAlgo {
14     public static void main(String[] args) {
15         String sDataToEncrypt = new String();
16         String sCipherText = new String();
17         String sDecryptedText = new String();
18         try{        /**
19         *  Step 1. Generate a DES key using KeyGenerator
20          */
21     KeyGenerator kyGn = KeyGenerator.getInstance("DES");
22     SecretKey scrtkey = kyGn.generateKey();
23     /**
24      *  Step2. Create a Cipher by specifying the following parameters
25      */
26     Cipher cipherDES = Cipher.getInstance("DES/CBC/PKCS5Padding");
27     /**
28     *  Step 3. Initialize the Cipher for Encryption
29     */
30     cipherDES.init(Cipher.ENCRYPT_MODE,scrtkey);
```

```
31     /***
32  Step 4. Encrypt the Data**/
33  sDataToEncrypt = "Hello World of Encryption using DES ";
34  byte[] bDataToEncrypt = sDataToEncrypt.getBytes();
35  byte[] bCipherText = cipherDES.doFinal(bDataToEncrypt);
36  sCipherText = new BASE64Encoder().encode(bCipherText);
37  System.out.println("Cipher Text generated using DES with CBC mode and PKCS5 Padding
38  cipherDES.init(Cipher.DECRYPT_MODE,scrtkey,cipherDES.getParameters());
39  //cipherDES.init(Cipher.DECRYPT_MODE,scrtkey);
40  byte[] bDecryptedText = cipherDES.doFinal(bCipherText);
41  sDecryptedText = new String(bDecryptedText);
42  System.out.println(" Decrypted Text message is " +sDecryptedText);
43  }catch (NoSuchAlgorithmException ae)
44  {System.out.println(" No Such Algorithm exists " + ae);
45  }catch (NoSuchPaddingException pe)
46  {System.out.println(" No Such Padding exists " +pe);
47  }
48          catch (InvalidKeyException ike)
49          {       System.out.println(" Invalid Key " + ike);
50          }       catch (BadPaddingException bp)
51          {       System.out.println(" Bad Padding " + bp);
52          }       catch (IllegalBlockSizeException ibs)
53          {       System.out.println(" Illegal Block Size " + ibs);
54          }       catch (InvalidAlgorithmParameterException pm)
55          {       System.out.println(" Invalid Parameter " + pm);
56          }       }       }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Dos and Don'ts: Avoid using Statistical PRNG, Inadequate Padding and Insufficient Key Size

2. This code uses statistical PRNG to create a URL for a receipt which remains active for a certain period of time after purchase

JoseImpl.java

```
103        String GenerateReceiptURL(String base_Url)
104        {
105            Random randomGen = new Random();
106            randomGen.setSeed((new Date()).getTime());
107            return(base_Url + randomGen.nextInt(400000000) + ".html");
108        }
```

3. Don't use inadequate/inappropriate RSA padding

JoseImpl.java

```
123        public Cipher getRSACipher() {
124            Cipher rsa_algo = null;
125            try {
126            rsa_algo =
127            javax.crypto.Cipher.getInstance("RSA/NONE/NoPadding");
128            }
129            catch (java.security.NoSuchAlgorithmException ae) {
130            log("this should never happen", ae);}
131            catch (javax.crypto.NoSuchPaddingException pe) {
132            log("this should never happen", pe); }
133            return rsa_algo; }
```

4. Don't use insufficient/inappropriate key size

.java

```
public static KeyPair getRSAKey() throws NoSuchAlgorithmException {
    KeyPairGenerator kyGn = KeyPairGenerator.getInstance("RSA");
    kyGn.initialize(512);
    KeyPair kp = kyGn.generateKeyPair();
    return kp;
    }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Dos and Don'ts: Implement Strong Entropy

1. The following code uses strong entropy for key generation using Java cryptographic extensions

```
JoseImpl.java
149
150
151     public static void main(String[] args) {
152         try {
153         // Initialize a secure random number generator
154         SecureRandom secRan = SecureRandom.getInstance("SHA1PRNG");
155         // Method 1 - Calling nextBytes method to generate Random Bytes
156         byte[] b = new byte[512];
157
158         secRan.nextBytes(b);
159         // Printing the SecureRandom number by calling secRan.nextDouble()
160         System.out.println("Secure Random num generated by calling nextBytes() is " + secRan.nextDouble());
161         // Method 2 - Using setSeed(byte[]) to reseed a Random object
162         int ByteCount_seed = 10;
163         byte[] bseed = secRan.generateSeed(ByteCount_seed);
164          System.out.println(" Seed value is " + new BASE64Encoder().encode(bseed));
165         secRan.setSeed(bseed);
166         System.out.println(" Secure Random num generated using setSeed(byte[]) is  " +
167         secRan.nextDouble());
168         } catch (NoSuchAlgorithmException ae)
169
170         System.out.println(" No Such Algorithm exists " + ae);
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Dos and Don'ts: Implement Strong Algorithms

2. The following code uses strong algorithm using Java cryptographic extensions

```
AES.java
1  package com.domo;
2
3  import javax.crypto.KeyGenerator;
4  import javax.crypto.SecretKey;
5  import javax.crypto.Cipher;
6  import java.security.NoSuchAlgorithmException;
7  import java.security.InvalidKeyException;
8  import java.security.InvalidAlgorithmParameterException;
9  import javax.crypto.NoSuchPaddingException;
10 import javax.crypto.BadPaddingException;
11 import javax.crypto.IllegalBlockSizeException;
12 import sun.misc.BASE64Encoder;
13 public class AES {
14 public static void main(String[] args) {
15 String sDataToEncrypt = new String();
16 String sCipherText = new String();
17 String sDecryptedText = new String();
18 try{
19 /**
20 *  Step 1. Generate an AES key using KeyGenerator
21 *  Initialize the keysize to 128 */
22      KeyGenerator kyGn = KeyGenerator.getInstance("AES");
23      kyGn.init(128);
24      SecretKey scrtkey = kyGn.generateKey();
25      /**
26      *  Step2. Create a Cipher by specifying the following parameters
27      */
28      Cipher aesCipher = Cipher.getInstance("AES");
29      /**
30      * Step 3. Initialize the Cipher for Encryption */
31      aesCipher.init(Cipher.ENCRYPT_MODE,scrtkey);
32      /**
33      *  Step 4. Encrypt the Data */
34      sDataToEncrypt = "Hello World of Encryption using AES ";
35      byte[] bDataToEncrypt = sDataToEncrypt.getBytes();
36      byte[] bCipherText = aesCipher.doFinal(bDataToEncrypt);
37      sCipherText = new BASE64Encoder().encode(bCipherText);
38      System.out.println("Cipher Text generated using AES is " +sCipherText);
```

```
39      /**
40      *  Step 5. Decrypt the Data
41      */
42      aesCipher.init(Cipher.DECRYPT_MODE,scrtkey,aesCipher.getParameters());
43      byte[] bDecryptedText = aesCipher.doFinal(bCipherText);
44      sDecryptedText = new String(bDecryptedText);
45      System.out.println(" Decrypted Text message is " +sDecryptedText);
46      }
47      catch (NoSuchAlgorithmException ae)
48      {
49      System.out.println(" No Such Algorithm exists " + ae);
50      }
51      catch (NoSuchPaddingException pe)
52      {
53      System.out.println(" No Such Padding exists " + pe);
54      }
55      catch (InvalidKeyException ike)
56      {
57      System.out.println(" Invalid Key " + ike);
58      }
59      catch (BadPaddingException bp)
60      {
61      System.out.println(" Bad Padding " + bp);
62      }
63      catch (IllegalBlockSizeException ibs)
64      {
65      System.out.println(" Illegal Block Size " + ibs); }
66      catch (InvalidAlgorithmParameterException pm)
67      {
68      System.out.println(" Invalid Parameter " + pm);
69      }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

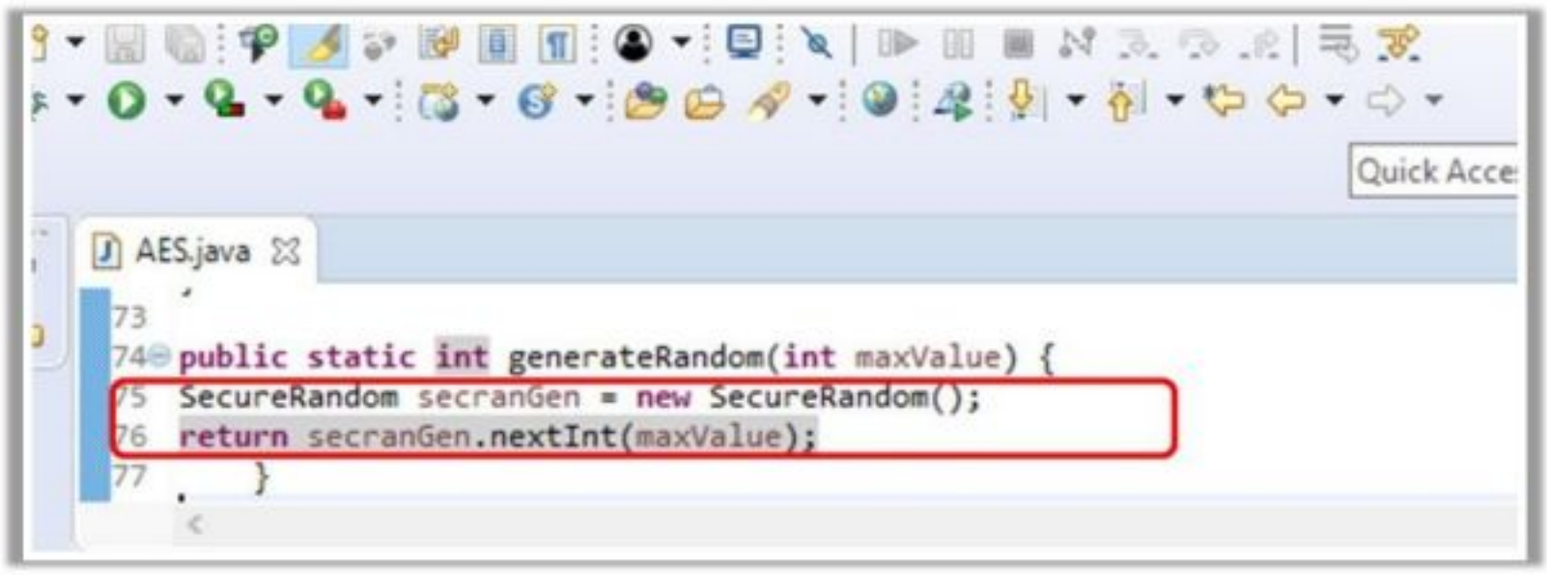
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Dos and Don'ts: Implement Strong Algorithms (Cont'd)

3. Use Java's SecureRandom class

- This code uses Java's **SecureRandom** class to generate cryptographically strong pseudo-random number



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Best Practices for Java Cryptography

- ✓ Securely manage **keys** that are used for code **signing** and **sealing**
- ✓ Use existing **crypto libraries** rather than creating your own cryptographic protocols
- ✗ Don't use **pseudorandom** number generators instead use cryptographically secured **random numbers**
- ✗ Don't store **encrypted keys** and **passwords** that are dependent on **garbage collector** in memory; rather use configuration files with authorized entities
- ✓ Securely **manage objects** that are transmitted across the **network** with signing and sealing
- ✓ Ensure that the **key length** used by algorithms is at least **128 bits** (Eg, RSA key 512 bits)
- ✗ Don't store **encryption keys**, keystore name, the alias and **password details** in the source code
- ✗ Don't store **encryption keys**, keystore name, password, alias, etc., in **unprotected external files**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Module Summary

- Cryptography deals with security issues in accordance with privacy, integrity, authentication and nonrepudiation
- Java platform offers cryptographic operations using APIs such as Java Cryptography Architecture and Java Cryptography Extension
- The Cipher Class in `javax.crypto` package performs the main functionality of cryptographic cipher using encryption and decryption
- JCA provides a specific framework for digital signatures with `java.security.Signature` class that provides functionality of signing and verifying digital signatures
- Digital certificate includes User (entity) Information, User's public key, Digital signature of the CA, Issue and expiry date
- Java Archive (JAR) is a file format that contains multiple files .i.e., class files and subsidiary resources associated with applets and applications
- Code signing is a security mechanism that performs digital signing of Java scripts and executables using cryptography algorithms to prevent malicious activities

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

JAVA

C | ASE

Certified | Application Security Engineer



Module 07

Secure Coding Practices for Session Management

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

Module Objectives

1. Introduction to Session Management in Java
2. Discuss Session Management in Spring Framework
3. Discuss Session Vulnerabilities and their Mitigation Techniques
4. Learn Best Practices and Guidelines for Secure Session Management

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Session Management

A session can be defined as a number of **requests** made by a particular client over a period of time

Session management can be used to keep track of the information of a **web user** in a session, such as number of requests etc.

Java session management is helpful in **storing** the information of user, application security and for timing out a session

Improper authentication and session management results in disclosure of the users' identity by stealing passwords, keys and session tokens

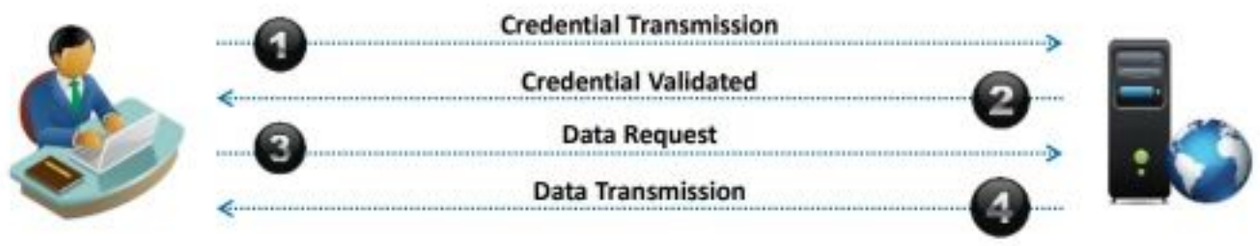
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Session Tracking

1. A session can be defined as a conversation between a **server** and a **client**
2. When there is a series of continuous requests and responses from the same client to a server, the server cannot identify from which client it is getting requests as **HTTP is a stateless protocol**
3. When there is a need to maintain the conversational state, **session tracking is needed**
4. Different methods of session tracking are **Cookies**, **URL Rewriting**, **Hidden Fields**, **Session Objects** etc.



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Session Tracking Methods: HttpSession

- HttpSession helps to store the full session object for a specific client
- HttpSession provides some methods which help to store attributes and remove attributes from a session

The following code shows how to create a new HttpSession object

SessionTracking.java

```
35
36
37  HttpSession session=request.getSession();
38  session.setAttribute("user", uname);
```

- This code creates a **session object** with a name of "session" and an attribute user with value name
- The session can be invalidated with the `void invalidate()` method

List of the HttpSession methods to help manage session

- **long getCreationTime():** It will return session creation time in milliseconds
- **String getId():** It will return the unique identifier of the session
- **long getLastAccessedTime():** Returns the time of the last request for the session
- **int getMaxInactiveInterval():** Returns the session active time
- **void invalidate():** This method will destroy the session
- **boolean isNew():** Returns value true or false of session creation
- **void setMaxInactiveInterval(int interval):** Active time of session in container

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Session Tracking Methods: Cookies

- A cookie, also known as an **HTTP cookie**, **web cookie**, or **browser cookie**, is a piece of data stored by a website within a browser, and then subsequently sent back to the same website by the browser
- Cookies were designed to be a **reliable mechanism** for websites to remember things that a browser had done there in the past

The following code shows how to create a new cookie

SessionTracking.java

```
42  Cookie user = new Cookie("username","Jummy");
43  user.setMaxAge(3600);
44  response.addCookie(user);
```

- This code creates a cookie with a name of "username" and a value of "Jummy"
- The cookie's expiration date is set with the **`setMaxAge()`** method to 3,600 seconds from the time the browser receives the cookie

The following code demonstrates how you would retrieve the value for a specific cookie

SessionTracking.java

```
46  String user1 = "";
47  Cookie[] ck = request.getCookies();
48  if (ck != null) {
49  for (int x = 0; x < ck.length; x++) {
50  if (ck[x].getName().equals("user"))
51          user1 = ck[x].getValue();
52      }
```

- In this code, an array of cookies is retrieved from the **HttpServletRequest** object using the **`getCookies()`** method
- The array is walked through until a cookie with the name of "user" is returned by the **`getName()`** method
- Once the cookie is found, the **`getValue()`** method is called to retrieve the value of the cookie

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Setting a Limited Time Period for Session Expiration

- Configure **`session-timeout`** attribute for setting session duration
- If the time period of session is set for a shorter duration, then the attacker gets less time to steal the **cookies,** thus reducing the risk of **stolen cookies**
- To implement session-timeout in JAVA configuration use **javax.servlet.http.HttpSession** API

Vulnerable Code

SessionTracking.java | web.xml

```
14  <session-config>
15  <session-timeout>60</session-timeout>
16    </session-config>
17  </web-app>
```

Design | Source

Secure Code

SessionTracking.java | web.xml

```
14  <session-config>
15  <session-timeout>10</session-timeout>
16    </session-config>
17  </web-app>
18
```

Design | Source

Setting Session Duration using JAVA Configuration

Vulnerable

SessionTracking.java

```
38  HttpSession session=request.getSession();
39  session.setAttribute("user", uname);
40  session.setMaxInactiveInterval(60*60);
```

Secure

SessionTracking.java

```
38  HttpSession session=request.getSession();
39  session.setAttribute("user", uname);
40  session.setMaxInactiveInterval(10*60);
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

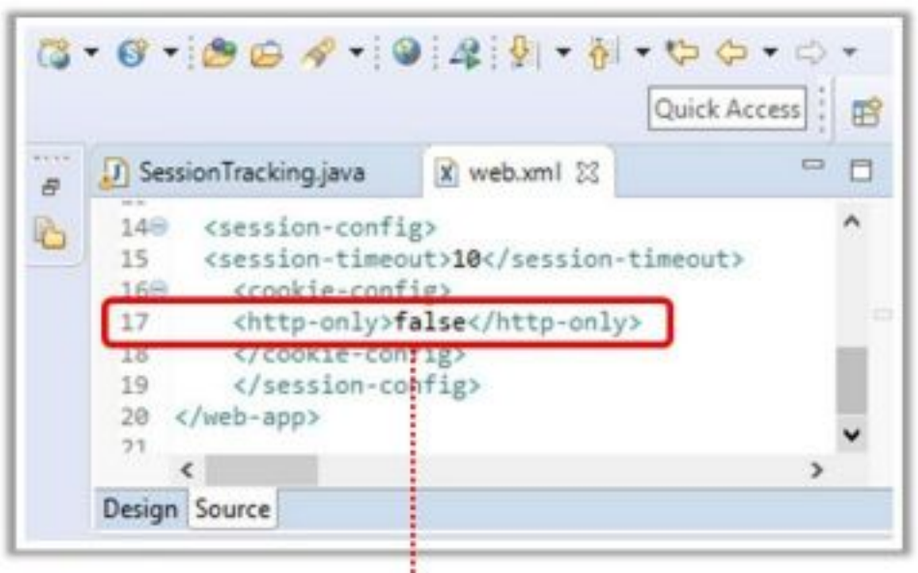
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Preventing Session Cookies from Client-side Scripts Attacks

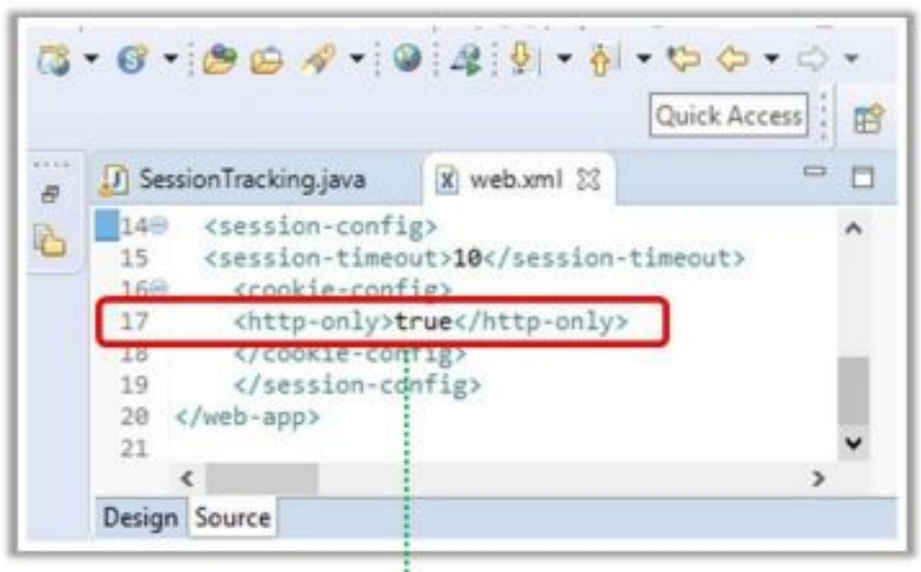
- To prevent the client script from accessing the session cookie, set the `httpOnlyCookie` attribute to `true` in the **web.xml** file

Vulnerable Code



- Setting `http-only` to `false`, then cookie is accessible from client script

Secure Code



- Setting `http-only` to `true`, then cookie is not accessible from client script

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Session Tracking Methods: URL Rewriting

I. URL rewriting is used where either a **browser** or a **wireless device** doesn't accept cookies. In this case, session tracking isn't possible by using cookies

II. When the web server detects that the browser is not accepting cookies then the session ID is **encoded** into the hyperlinks on the **webpages**

III. Web server **extracts** the **ID** from the **URL address** and traces out the HTTP session using the **getSession()** method

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Example Code for URL Rewriting

PostServlet.java

```
39    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Se
40        PrintWriter out = response.getWriter();
41        String Username = request.getParameter("user");
42        String Password = request.getParameter("passwd");
43        out.println("<html>");
44        out.println("<head><title>Post Servlet Example</title></head>");
45        out.println("<body><h1>");
46        if (Username.equals("Krish") && Password.equals("Nalini")) {
47        out.println("Welcome " + Username +"!<br/><br/>");
48        HttpSession Hs = request.getSession();
49        Hs.setAttribute("Username",Username);
50          out.println("<a href=\"linkedServlet?username="+Username+">Click here</a>");
51        } else {
52        out.println("Invalid username/Password. Go back and try again.");
53        }
54        out.println("</h1></body></html>");
55        out.close();
56
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Session Tracking Methods: Hidden Fields

- Hidden form fields are basically used for session maintenance and also used for **tracking sessions**
- It supports almost all the browsers and does not need any special **server requirements**
- Limitation of the hidden field is, it works only for a sequence of dynamically **generated forms**

MyShopingCart.java

```
31    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws Ser
32        response.setContentType("text/html");
33        PrintWriter out = response.getWriter();
34        out.println("<HEAD><TITLE>Shopping Cart  Present</TITLE></HEAD>");
35        out.println("<BODY>");
36        // Cart objects are being passed as the item parameter.
37        String[] objects = request.getParameterValues("item");
38        // Print the current cart objects.
39        out.println("You currently have the following objects in your cart:<BR>");
40        if (objects == null) {
41        out.println("<B>None</B>");
42        }
43        else {
44        out.println("<UL>");
45        for (int a = 0; a < objects.length; a++) {
46        out.println("<LI>" + objects[a]);
47        }
48        out.println("</UL>");
49        }
50        // Ask if the user wants to add more objects or check out.
51        // Include the current objects as hidden fields so they'll be passed on.
52        out.println("<FORM ACTION=\"/servlet/ShoppingCart\" METHOD=POST>");
53        if (objects != null) {
54        for (int a = 0; a < objects.length; a++) {
55        out.println("<INPUT TYPE=hidden NAME=object VALUE=\"" +
56        objects[a] + "\">");
57        } }
58        out.println("Would you like to<BR>");
59        out.println("<INPUT TYPE=submit VALUE=\" Add More Items \">");
60        out.println("<INPUT TYPE=submit VALUE=\" Check Out \">");
61        out.println("</FORM>");
62        out.println("</BODY></HTML>");
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Session Tracking Methods: Session Objects

1. Session object is used for the maintenance of a user session related information on the **server side**
2. Session object is used to **store**, **retrieve** and **remove information** based on program logic
3. Each session object created for a user persists until the user **logs out** or the user remains idle for the **session expiration time**
4. Basically on an average a typical session lasts for **10 minutes** by **default**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Session Management in Spring Framework

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Spring Session Management

- **The following options can be used to implement an `HttpSession` in spring applications**

1. **Injecting HttpSession**
   - Inject **HttpSession** wherever required
   - Retrieving and saving data in the session has to be managed **manually**

2. **Accept it as a parameter**
   - Useful for **web tire applications**

Example: Injecting HttpSession

ShopingCartService.java

```
4
5  import org.springframework.beans.factory.annotation.Autow
6  import org.springframework.stereotype.Service;
7
8  @Service
9  public class ShopingCartService {
10 @Autowired HttpSession httpsesion;
```

Example: Accept as Parameter

MyShoppingCartController.java

```
1  package com.myproject.webapp;
2
3  import javax.servlet.http.HttpSession;
4
5  import org.springframework.stereotype.Controller;
6  import org.springframework.web.bind.annotation.RequestMapping;
7
8  @Controller public class MyShoppingCartController {
9      @RequestMapping("/addToMyCart")
10
11     public String addCart(long prodId, HttpSession httpSession) {
12
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Spring Session Management (Cont'd)

3. **Creating Bean with session scope**
   - Retrieving and saving changes to the Bean is taken care by spring
   - Spring lifecycle methods(**@PostConstruct** or **@PreDestroy** annotated methods) defined in the Bean will be implemented automatically

4. **Using @SessionAttributes annotation**
   - Include **@SessionAttributes** annotation for model attributes to be stored in the session
   - **@SessionAttributes** copies the specified model attributes to HttpSession before rendering the view

Example: Creating Session Scope Bean

MyShoppingCartService.java

```
4  import org.springframework.context.annotation.Scope;
5  import org.springframework.context.annotation.ScopedProxyMode;
6  import org.springframework.stereotype.Component;
7
8  @Component
9  @Scope(proxyMode=ScopedProxyMode.TARGET_CLASS, value="session")
10 public class MyShoppingCartService {
```

Example: Using @SessionAttribute

MyShoppingCartController.java

```
15 @SessionAttributes("myRequestObject")
16 public class MyShoppingCartController {
17     @RequestMapping("/addToMyCart")
18
19     public String addCart(long prodId, HttpSession httpSession)
20     {
21
22         return null;
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Session Management using Spring Security

- To control session creation using spring security, configure create-session attribute with the following options
  - **Always:** Always creates a **session** if it does not exist
  - **ifRequired (Default)**: Creates session when required
  - **Never:** Spring security will not create sessions but will use **session created by application**
  - **Stateless:** Application will not **create** or **use session**

Example : JAVA Configuration

```
20 }
21 protected void configure(HttpSecurity http) throws Exception
22 {
23     http.sessionManagement().sessionCreationPolicy(SessionCreatio
24 }
25 }
```

Example : XML Configuration

```
11
12 <http create-session="ifRequired"></http>
13
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Restricting Concurrent Sessions per User using Spring Security

- Concurrent sessions for each user can be controlled using **Spring Security**

Vulnerable Code

```
11
12 <http create-session="ifRequired">
13 <session-management>
14 <concurrency-control max-sessions="5"/>
15  </session-management>
16 </http>
```

- Enables attackers to **login** at the same time as the user and perform attacks

Secure Code

```
11
12 <http create-session="ifRequired">
13 <session-management>
14 concurrency-control max-sessions="1"/>
15  </session-management>
16 </http>
```

- Prevent attackers to **perform attacks** when a user is logged in

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Restricting Concurrent Sessions per User using Spring Security (Cont'd)

- To enable concurrent **session-control**
- Enabling concurrent sessions makes sure to notify the spring security session registry that the session is destroyed

Enabling Concurrent Session-control

- Include the following listener in the web.xml

```
8<listener>
9<listener-class>
0org.springframework.security.web.session.HttpSessionEventPublisher
1</listener-class>
2</listener>
13
```

Example : Control Concurrent Sessions JAVA Configuration

```
25 protected void configure(HttpSecurity http) throws Excepti
26 {
27     http.sessionManagement().maximumSessions(2);
28 }
```

Example: Bean for enabling Concurrent Session-control

```
@Bean
public HttpSessionEventPublisher httpSessionEventPublisher() {

    return new HttpSessionEventPublisher();
}
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Controlling Session Timeout

```
9     http://www.springframework.org/schema/beans/spring-beans-4.2.xsd">
10
12 <http create-session="ifRequired">
13 <session-management>
14 <concurrency-control max-sessions="1" expired-url="UserLogin.html"/>
15 </session-management>
16 </http>
```

Redirecting user to a configured URL on sending a request with an expired session id

Configuring session-fixation-protection in XML

```
9     http://www.springframework.org/schema/beans/spring-beans-4.2.xsd">
10
12 <session-management  invalid-session-url="login.html">
13 <concurrency-control max-sessions="1" expired-url="UserLogin.html"  />
14 </session-management>
```

Source | Namespaces | Overview | beans | sec

Redirecting user to a configured URL on sending a request with an Invalid session id

```
22 }
23 @Override
24 protected void configure(HttpSecurity http) throws Exception
25 {
26     http.sessionManagement()
27     .invalidSessionUrl("Relogin.html")
28     .maximumSessions(1).expiredUrl("UserLogin.html");
```

Java configuration for redirecting user to a configured URL on requesting a page with expired / invalid session id

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

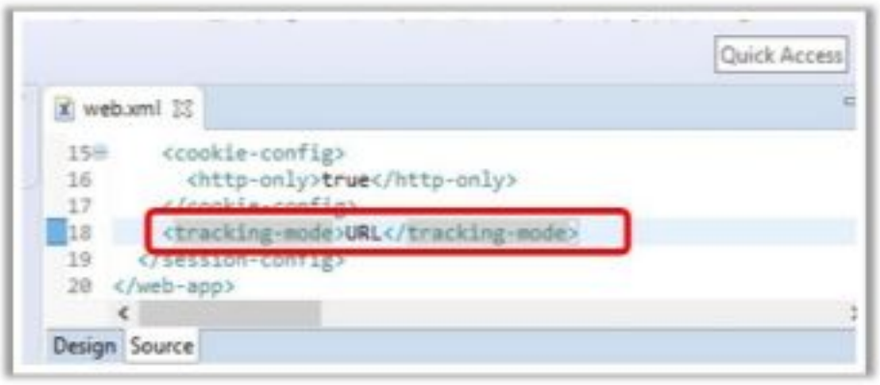
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Prevent using URL Parameters for Session Tracking

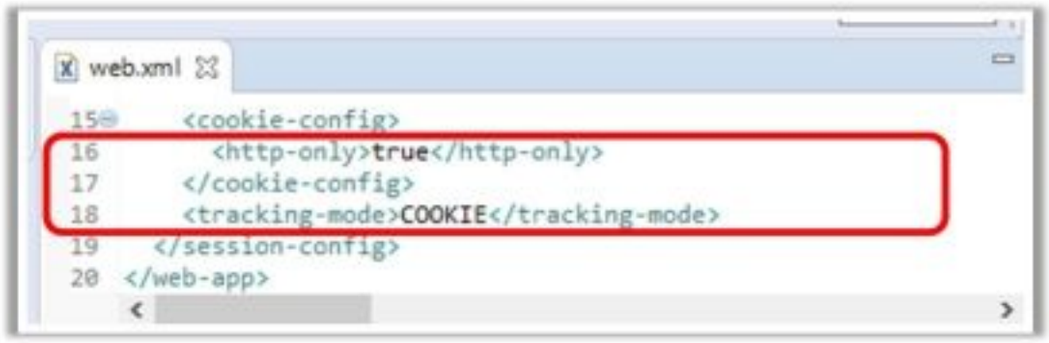
- A URL with a session tracking parameter can be easily used to steal the session
- Disable URL rewriting to prevent appending **JESSIONID** to the URL
- Set `disable-url-rewriting="true"` in the `<http>` namespace (Servlet 3.0 onwards)

Vulnerable Code

- Enables attackers to login at the same time as the user and perform attacks

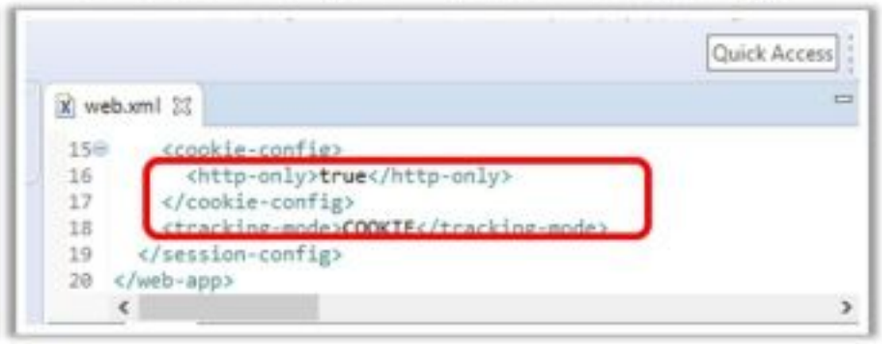


Configuring Session Fixation Protection in web.xml



Secure Code

- Prevents attackers from performing attacks when a user is logged in



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Prevent Session Fixation with Spring Security

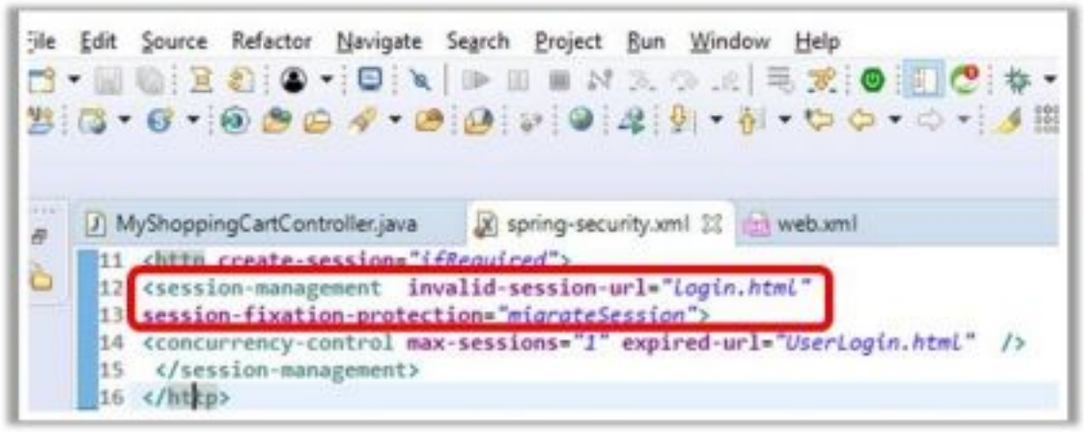
Spring Framework provides Session Fixation Protection

- Configure **session fixation protection** attribute to prevent session fixation
- **Session fixation protection** attribute determines the behavior of the existing session when an already authenticated user tries to login again

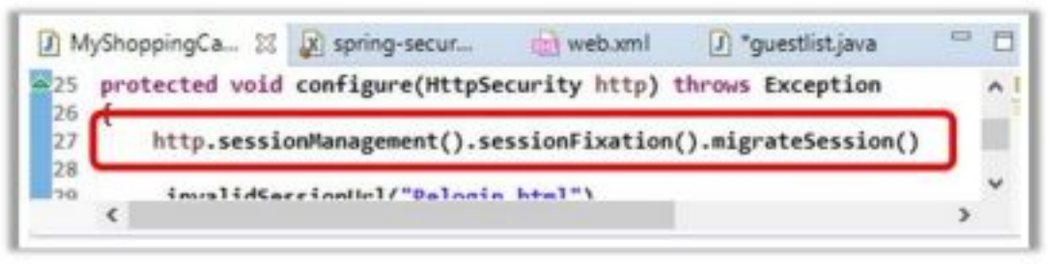
The following values can be assigned:

- `migrateSession-(Default)`: Creates a new http session and invalidates the old session after copying the attributes of the old session
- `None`: Old session remains and is not invalidated
- `newSession`: New http session is created without copying the attributes of the old session

Configuring Session Fixation Protection in XML



JAVA Configuring for Session Fixation Protection



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Prevent Session Fixation with Spring Security (Cont'd)

Vulnerable Code

- Enables attackers to login at the same time as the user and perform attacks

MyShoppingCartController.java | spring-security.xml | web.xml

```
11 <http create-session="ifRequired">
12 <session-management  session-fixation-protection="none">
13 <concurrency-control max-sessions="1" expired-url="UserLogin.html"
14  </session-management>
15 </http>
```

Secure Code

- Prevents attackers from performing attacks when a user is logged in

MyShoppingCartController.java | spring-security.xml | web.xml

```
11 <http create-session="ifRequired">
12 <session-management  invalid-session-url="login.html"
13 session-fixation-protection="migrateSession">
14 <concurrency-control max-sessions="1" expired-url="UserLogin.html"  />
15  </session-management>
16 </http>
17
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Use SSL for Secure Connection

- Always configure SSL for web applications in order to protect session IDs from prying eyes
- Set the `SSLEnable="true"` element to true in the **server.xml** file

Implement SSL in server.xml

MyShoppingCartController.java | spring-security.xml | server.xml

```
91
92    <Connector
93 clientAuth="true" port="8443" minSpareThreads="5" maxSpareThreads="75"
94 enableLookups="true" disableUploadTimeout="true"
96 scheme="https" secure="true" SSLEnabled="true"
98 keystoreType="JKS" keystorePass="password"
99 truststoreFile="/Users/mporges/Desktop/tomcat-ssl/final/server.jks"
101 SSLVerifyClient="require" SSLEngine=
102 />
103
```

Implement SSL in spring-security.xml by setting force Https property to true

spring-security.xml

```
48 <beans:bean id="authenticationProcessingFilterEntryPoint" class="org.springframe
49 <beans:constructor-arg name="strength" value="12" />
50 <beans:property name="forceHttps" value="true"></beans:property>
51 <beans:property name="loginUrl" value="true"></beans:property>
52  </beans:bean>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Session Vulnerabilities and their Mitigation Techniques

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Session Vulnerabilities

- Session hijacking includes exploiting the **web session** control mechanism that is usually managed by the **session token**
- In a **session hijack attack**, the attacker compromises a session by obtaining a session token either through stealing or predicting and gains access to the webserver
- Different ways of compromising a session token are:
  - Predictable session token
  - Session sniffing
  - Client-side attack (XSS, malicious codes, trojans)
  - Man-in-the-middle attack
  - Man-in-the-browser attack

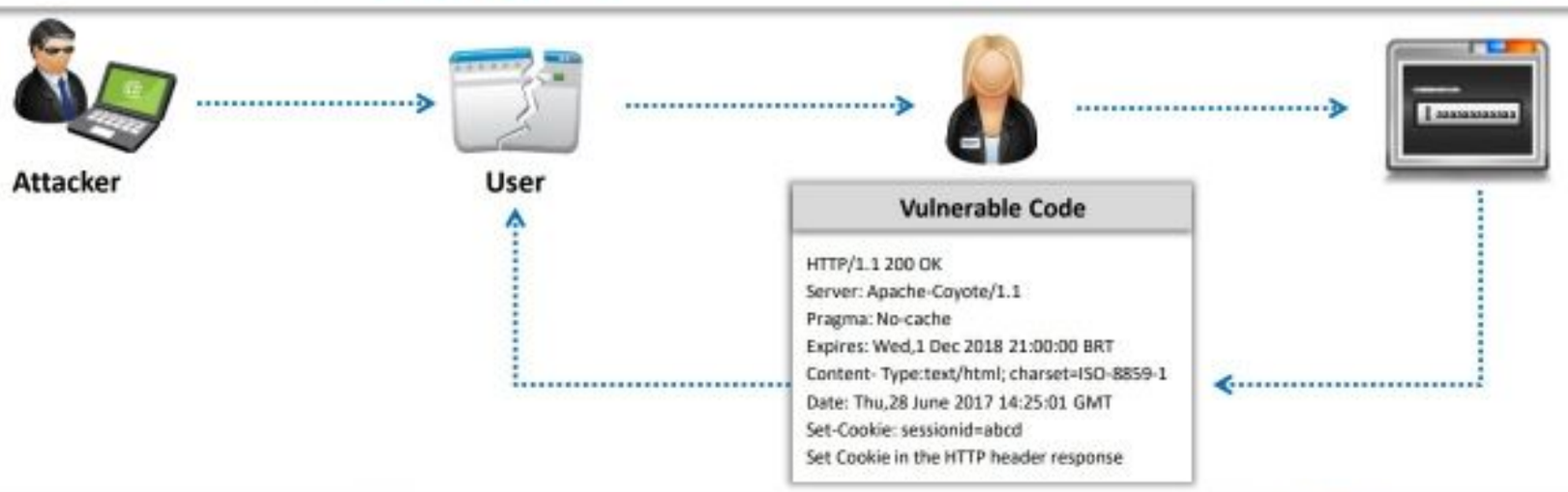
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Types of Session Hijacking Attacks

1. **Attack using client-side scripting**
   - Attackers can fix the session ID value by changing the victim's cookie and inserting a **JavaScript code** in the **URL** that will be executed in the victim's browser
   - http://website.kom/<script>document.cookie="sessionid=abcd";</script>
2. **Attack using <META> tag**
   - Attackers use **code injection attacks** by injecting **malicious code** into the URL and send it to the victim
   - http://website.kon/<meta http-equiv=Set-Cookie content="sessionid=abcd">
3. **Attack through HTTP header response**
   - An attacker creates a **cookie** value of the Session ID, **modifies** the **server response** and **intercepts** the **packages** exchanged between the client and the web application by inserting the **Set-Cookie parameter**



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Types of Session Hijacking Attacks (Cont'd)

4. **Session fixation**
   - The session fixation attack is a kind of session hijacking, which **steals** the **established session** between the client and the web server once the user logs in
   - The session fixation attack fixes an established session on the **victim's browser**, to begin attack before the user logs in



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Types of Session Hijacking Attacks (Cont'd)

5. Stealing session ID through HTTP GET request

- Attackers embed the session ID information in the URL and it is received by the application through **HTTP GET requests** when the client clicks on the **links**
- **Example**: http://wxyz.com/wxyz.jsp?article=1234;session id=ER69045454

Stealing session ID using HTTP POST command

```
FORM METHOD=POST ACTION=?/cgi-bin/news.pl?>
<INPUT TYPE =?hidden? Name =?sessionid?
VALUE=?ER69045454?>
<INPUT TYPE =?hidden? Name =?allowed? VALUE=?true?>
<INPUT TYPE =?submit? Name =?Read News Article?>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Countermeasures for Session Hijacking

- Regularly clear the **history** and **offline content**
- See that session is **expired** after users log out
- Make sure that **cookies** and **sessions** are removed from the browser
- **Life span** of the session or a cookie should be reduced
- Always implement **https** over http
- Clicking on **unknown links** should be avoided
- Session key has to be a **long random number** or **string**, so that it avoids guessing of a **valid session** and even avoids the chances of a **brute force attack**
- **Session ID** has to be **regenerated** in order to **avoid session fixation** and it is not possible for the attacker

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Countermeasures for Session ID Protection

1. If implementing SSL is not possible due to a performance issue or any other reason then protect session IDs using other ways
2. Alter the Session IDs frequently to reduce the validity of a session ID
3. Do not include session IDs as part of URLs, as they can be **cached** by the **browser**, sent in the **referer header**, can be traced from the logs, or can be forwarded to a 'friend' by mistake
4. Session IDs should be **long**, **complicated**, **random numbers** that cannot be easily guessed
5. Modify the session IDs while **switching** to **SSL**, **authenticating**, or other **major transitions**. Never accept session IDs chosen by a **user**
6. Protect the entire session using **SSL**. This ensures that the session ID (e.g., session cookie) cannot be **stolen** using man-in-the-middle attack, which is a major threat to session IDs

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Best Practices and Guidelines for Secure Session Management

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Best Coding Practices for Session Management

1. Session Invalidate

- Session invalidate can be defined as removing the **HttpSession** object and its values from the system

Example of Session Invalidate

PostServlet.java

```
36  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletEx
37
38      response.setContentType("text/html");
39      HttpSession Hsession = request.getSession();
40      Hsession.removeAttribute(Login);
41      Hsession.removeAttribute(password);
42      Hsession.invalidate(); response.sendRedirect("http://ec.com/servlets/login.html");
43      }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Best Coding Practices for Session Management (Cont'd)

2. Session Validate

- Each session should have a defined **timeout** when there is no activity
- If needed, a session hard limit can also be used in some cases in order to **kill** a **session** irrespective of session activity
- Such settings can be done by either in **configuration** files or directly in the code itself

Session Validate Sample Code

web.xml

```
8     <welcome-file>default.html</welcome-file>
9     <welcome-file>default.htm</welcome-file>
10    <welcome-file>default.jsp</welcome-file>
11  </welcome-file-list>
12  <session-config>
13    <session-timeout>30</session-timeout>
14    <cookie-config>
15      <http-only>true</http-only>
```

Design | Source

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Best Coding Practices for Session Management (Cont'd)

3. Use POST not GET

- As a part of best practice it is always recommended to use POST instead of GET, as all the major browsers use GET by default for **form submission**
- In the HTML forms POST should be specified like this **<form method ="POST"...>**

Vulnerable Code

Login.jsp

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional/
4 <html>
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; charse
7 <title>Register</title>
8 </head>
9 <body>
10 <form action="LoginController" method="get"></form>
11 </body>
```

Secure Code

Login.jsp

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3
4 <html>
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7 <title>Register</title>
8 </head>
9 <body>
10 <form action="LoginController" method="Post"></form>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Best Coding Practices for Session Management (Cont'd)

4. Customize the Session Cookie Name, Path and Domain

- As a part of best practice it is always recommended to change the session cookie name from **JSESSIONID**
- This will protect the cookie from overwriting values from some other application

Vulnerable Code

MyShoppingCartController.java | spring-security.xml | web.xml

```
38 }
39 @Bean
40 public CookieSerializer cookieserializer()
41 {
42     DefaultCookieSerializer serializer = new DefaultCookieSerializer(
43     serializer.setCookieName("JSESSIONID");
44     serializer.setCookiePath("/");
45     serializer.setDomainNamePattern("^.+?\\.(\\w+\\.[a-z]+)$");
46     return serializer;
47
48 }
```

Secure Code

hoppingCartController.java | spring-security.xml | web.xml

```
Bean
ublic CookieSerializer cookieserializer()

    DefaultCookieSerializer serializer = new DefaultCookieSerializer();
    serializer.setCookieName("MYAppCookie");
    serializer.setCookiePath("/");
    serializer.setDomainNamePattern("^.+?\\.(\\w+\\.[a-z]+)$");
47  return serializer;
```

Configuring Default Cookie Name in web.xml

```
<session-descriptor><cookie-name>MYWEBAPIID<cookie-name></session-descriptor>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Best Coding Practices for Session Management (Cont'd)

JAVA CASE

5. Generate a High Entropy Session ID

- Attackers can guess the session ID if it is of low entropy which is **completely predictable**
- Generate a **complex** and **less** predictable session ID

Vulnerable Code

LowEntropy.java

```
1  package com.demo;
2
3  import java.math.BigInteger;
4  import java.security.SecureRandom;
5
6  public class LowEntropy {
7  public static int TOKEN_SIZE=2;
8  static SecureRandom SECURE_RANDOM=new SecureRandom();
9  byte[] token;
10 public void EntropicToken()
11 {
12     token=new byte[TOKEN_SIZE];
13     SECURE_RANDOM.nextBytes(token);
14     System.out.println(new BigInteger(token).toString(TOKEN_SIZE));
15 }
16 public static void main(String[] args) {
17     LowEntropy loE=new LowEntropy();
18     loE.EntropicToken();
19     }
```

- If the size of the random key generated and stored is small, it is easy to guess

Secure Code

LowEntropy.java

```
1  package com.demo;
2
3  import java.math.BigInteger;
4  import java.security.SecureRandom;
5
6  public class LowEntropy {
7  public static int TOKEN_SIZE=16;
8  static SecureRandom SECURE_RANDOM=new SecureRandom();
9  byte[] token;
10 public void EntropicToken()
11 {
12     token=new byte[TOKEN_SIZE];
13     SECURE_RANDOM.nextBytes(token);
14     System.out.println(new BigInteger(token).toString(TOKEN_SIZE));
15 }
16 public static void main(String[] args) {
17     LowEntropy loE=new LowEntropy();
18     loE.EntropicToken();
19     }
```

- This is a complex key which is not predictable

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Best Coding Practices for Session Management (Cont'd)

JAVA CASE

6. Do not Store Non-serializable Objects into a Session

- Non-serializable session objects can not be **replicated**
- For better reliability and performance it is recommended that **session data is serializable**

Vulnerable Code

BestClassInitialize.java

```
1  package com.demo;
2
3  import javax.servlet.http.HttpSession;
4
5  public class BestClassInitialize  {
6
7      String globName;
8      String globValue;
9
10     public void addToSession(HttpSession session)
11     {
12         session.setAttribute("glob", this);
13
14     }
```

Secure Code

BestClassInitialize.java

```
1  package com.demo;
2  import java.io.ObjectInputStream;
3  import java.io.ObjectOutputStream;
4
5  import java.io.Serializable;
6
7  import javax.servlet.http.HttpSession;
8
9  public class BestClassInitialize implements Serializable {
10     private static final long serialVersionUID = 1L;
11     String globName;
12     String globValue;
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Checklist to Secure Credentials and Session IDs

Following is the checklist to find out the flaws in user credentials and session IDs

Check security aspects of all the stored credentials using **hashing** or **encryption**
Check if the credentials can be **guessed** or **overwritten**
Check if the session IDs are at all vulnerable to attacks such as **session fixation** etc.
Check whether the session IDs are **exposed** in the **URL**
Check whether the passwords, session IDs and other credentials are sent via **TLS connections** or not
Check if SSL is used by the **login page**
Check if the audit is logged with **who**, **when**, **from**, **where** and **what data**
Check if **inbuilt session management** is used or not
Check whether the **new**, **preset** or **invalid session ids** are **rejected**
Check the security levels of **password reset** and **Question** and **answer clues**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Guidelines for Secured Session Management

Guidelines

Password Strength

- Ensure that the password is of a enough length and complexity, using **combinations** of alphabet, numbers and non alphanumeric characters

Password Use

- Users should be restricted by number of **login attempts**
- Avoid the display of the passwords provided during failed login attempts as there are chances of **exploitation**
- Ensure that the user is provided with all the **login details** along with **date** and **time**

Password Storage

- It is always better to store the passwords either in **hashed** or **encrypted** format to avoid exposure
- Even **decryption keys** should be strongly **protected**

Browser Caching

- Make sure that all the **authentication pages** are marked with **no cache tag** in order to prevent usage of previously typed user credentials
- Latest browsers have feature called **AUTOCOMPLETE=OFF flag** to prevent credentials and sensitive information in autocomplete caches

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Module Summary

- In Java, session tracking is done with the help of Cookies, URL Rewriting, Hidden Fields, Session Objects, etc.
- Improper session management results in disclosure of the users' identity by stealing passwords, keys and session tokens
- Attacker compromises a session by obtaining a session token either through stealing or predicting and gains access to the webserver
- Session IDs should be long, complicated, random numbers that cannot be easily guessed
- A URL with a session tracking parameter can be easily used to steal the session

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

JAVA

C | ASE

Certified | Application Security Engineer

Module 08

Secure Coding Practices for Error Handling

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

Module Objectives

1. Introduction to Exception and Error Handling in Java
2. Discuss Erroneous Exceptional Behaviors
3. Learn Dos and Don'ts in Error Handling
4. Discuss Spring MVC Error Handling
5. Discuss Exception Handling in Struts 2
6. Learn Best Practices for Error Handling
7. Introduction to Logging in Java
8. Discuss Logging using Log4j
9. Learn Secure Coding in Logging
10. Learn Best Practices for Logging

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Introduction to Exceptions

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Exception and Error Handling

- Exception and error occur when an **unexpected** or **abnormal event** takes place during the execution of the application
- In Java, exceptions and errors belong to the parent class **Throwable**
- An exception can be classified into two types namely, checked (**compile time**) exceptions and unchecked (**runtime**) exceptions

Checked Exceptions

- Belong to the base class **Exception**
- They occur when there is an **error** found in **code**
- These exceptions are identified by the **compiler** and can be handled by programmers

Unchecked Exceptions

- Belong to the base class **RuntimeException**
- They occur when an **error** arises during the execution of the **program**
- These errors cannot be handled by programmers

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Exception and Error Handling (Cont'd)

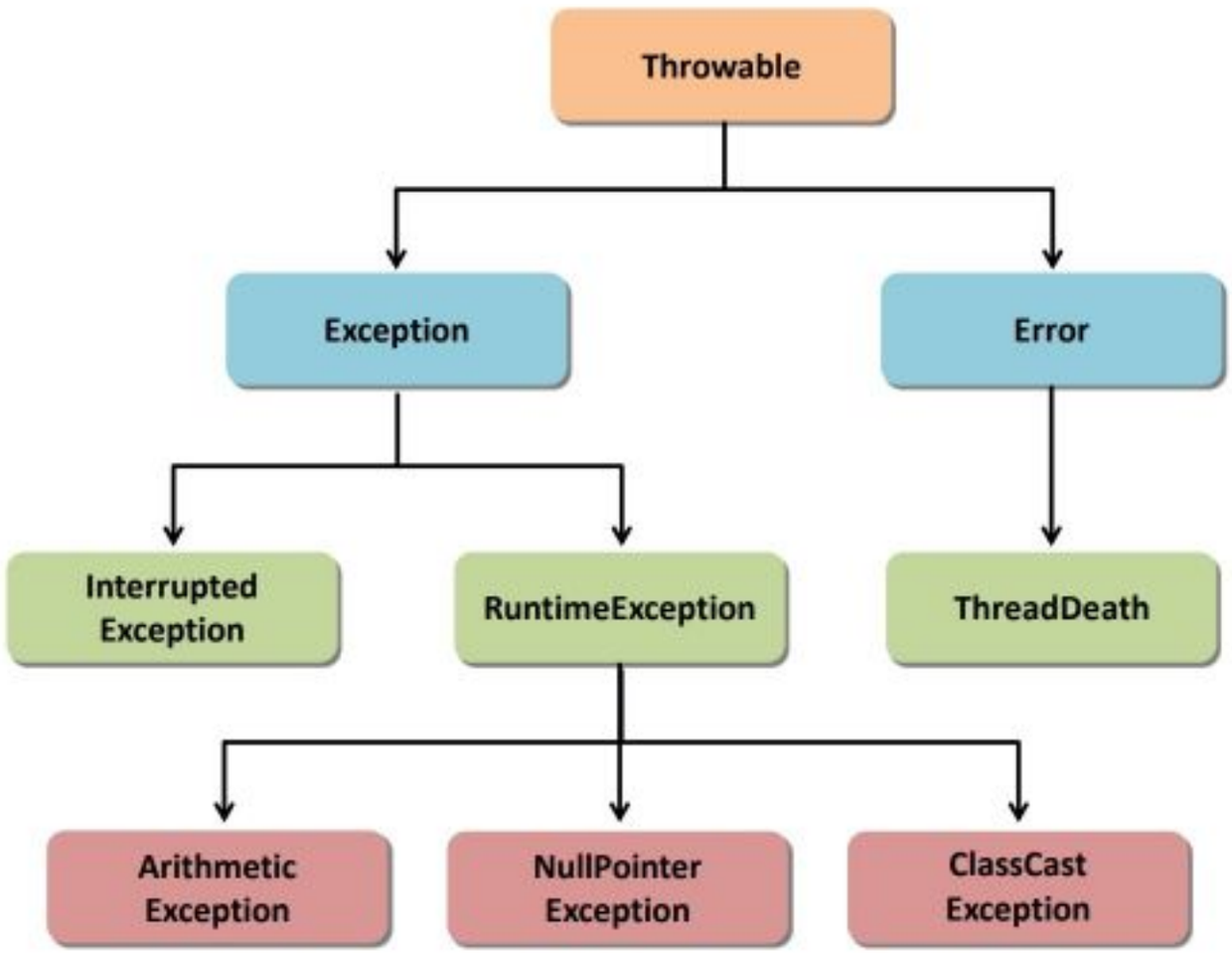
- Errors in Java are caused due to **irrecoverable conditions** such as memory leak, LinkageError, etc. that cannot be handled in programs
- Developers should ensure that exceptions are **properly handled** and do not disclose any information to the user/attacker
- Attackers can make use of information gained from error messages to perform **attacks**
- Information leakage can lead to **social engineering exploits**
- Unhandled exceptions may result in **abnormal termination** of the application

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

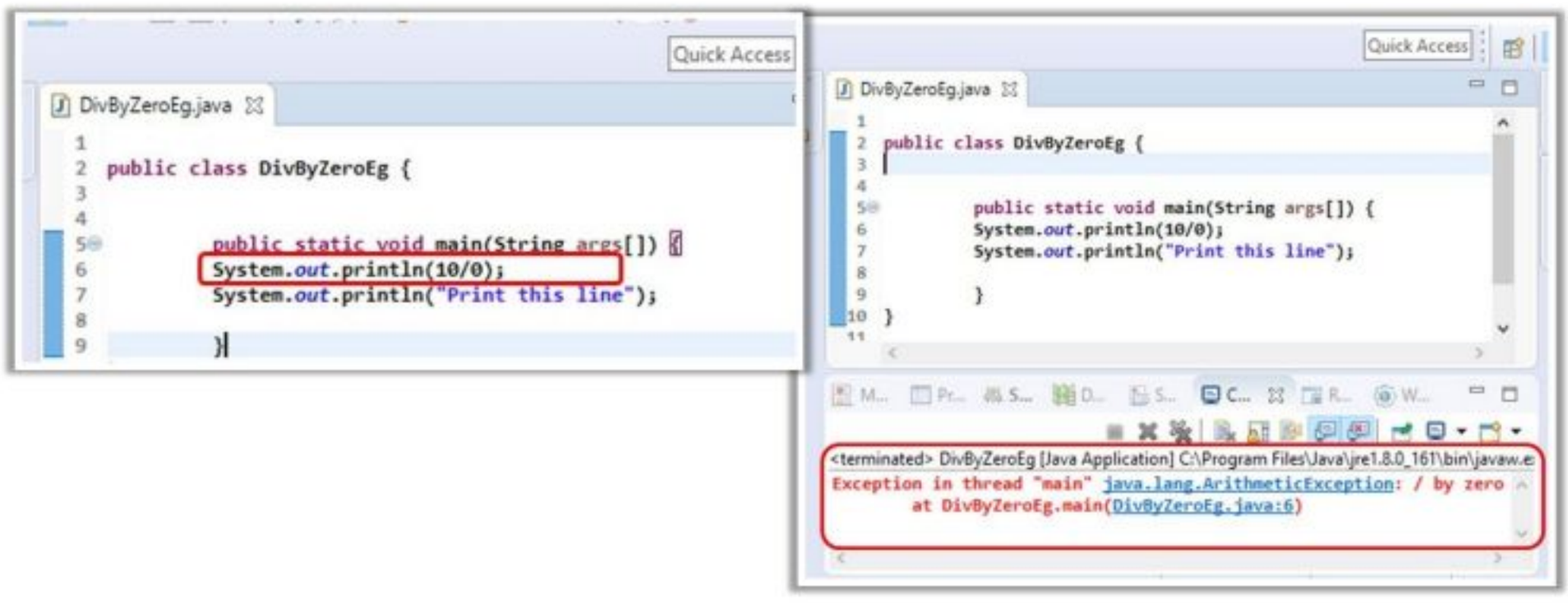
Exception and Error Handling (Cont'd)



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Example of an Exception

- Given below is a sample vulnerable code that throws an **arithmetic exception**
- An **exception message** that is thrown is shown below



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Handling Exceptions in Java

- **Exceptions in Java are handled using 'try-catch-finally' blocks**

Try

- The lines of code that may cause an exception are placed **inside** a try block
- If the try block encounters an exception, the remaining **statements** in the try block are **not executed**

Catch

- The exceptions that are thrown in the try block are **caught** and **handled** in the catch block

Finally

- The code in the finally block is always executed regardless of the **occurrence** of **exceptions**
- The finally block generally includes code that **frees resources, closes connections**, etc.

Syntax of try-catch-finally Block

DivByZeroEg.java

```
4
5       public static void main(String args[]) {
6           try {
7       System.out.println(10/0);
8       System.out.println("Print this line");
9           }
10          catch(Exception e)
11          {
12
13          }
14          finally {
15              System.exit(0);
16          }
17      }
18  }
19
20
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Handling Exceptions in Java (Cont'd)

- **Handling exceptions using 'try-with-resources' blocks**

	Ensures resources are closed automatically
	Doesn't require **finally** clause to close resources
	To use **try-with-resources** implement **AutoCloseable** (java.lang.AutoCloseable) interface

Example of try-with-resources Block

DivByZeroEg.java

```
13      private static void ReadFile() throws IOException {
14          try(FileInputStream input =
15          new FileInputStream("file.txt"))
16          {
17          int data = input.read();
18          while(data != -1){
19          System.out.print((char) data);
20          data = input.read(); }
21          } }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Handling Exceptions in Java (Cont'd)

try-with-resource vs try-catch-finally Blocks

try-with-resource Block	try-catch-finally Block
Resources are closed on completing the try block	Need to explicitly close the resources in the finally block
On encountering an error while closing the resource, the exception from the try block is thrown	On encountering an error while closing the resource from the finally block, the main relevant exception from the try block is hidden and the exception from the finally block is thrown

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Exception Classes Hierarchy

Throwable		
Error	1. LinkageError	2. VirtualMachineError
Exception	1. ClassNotFoundExeption	2. IllegalAccessException
	3. InstatntiationException	4. InterruptedException
	5. IOException	EOFException
		FileNotFoundException
	6. RuntimeException	ArithmeticException
		ArrayStoreException
		ClassCastException
		IllegalArgumentExeption
		IllegalMonitorStateException
		NegativeArraySizeException
		NullPointerException
		SecurityException

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Exceptions and Threats

Exception Name	Description of Information Leak or Threat
`java.io.FileNotFoundException`	Discloses file system structure, user name enumeration
`java.sql.SQLException`	Database structure, user name enumeration
`java.net.BindException`	Enumeration of open ports when untrusted client can choose server port
`java.util.ConcurrentModificationException`	May provide information about thread-unsafe code
`javax.naming.InsufficientResourcesException`	Insufficient server resources (may aid DoS)
`java.util.MissingResourceException`	Resource Enumeration
`java.util.jar.JarException`	Underlying file system structure
`java.security.acl.NotOwnerException`	Owner Enumeration
`java.lang.OutOfMemoryError`	Denial-of-Service
`java.lang.StackOverflowError`	Denial-of-Service

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Erroneous Exceptional Behaviors

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Erroneous Exceptional Behaviors

- Suppressing/ignoring checked exceptions
- Disclosing sensitive information
- Logging sensitive data
- Restoring objects to prior state, if a method fails
- Avoid using statements that suppress exceptions
- Prevent access to untrusted code that terminates JVM
- Never catch **NullPointerException**
- Never allow methods to throw **RuntimeException**, **Exception**, or **Throwable**
- Never throw undeclared checked exceptions
- Never let checked exceptions escape from **finally** block

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Erroneous Exceptional Behaviors: Suppressing or Ignoring Checked Exceptions

- **java.lang.InterruptedException** is thrown when a thread is interrupted while sleeping or waiting
- The run() method of **Runnable** interface cannot throw a checked exception but should handle **InterruptedException**

Vulnerable Code

- In the below example, code catches and suppresses **InterruptedException**
- The run()method caller **fails** to check that an interrupted exception has occurred

Demo.java

```
1
2  public class Demo implements Runnable {
3
4      @Override
5      public void run() {
6          try {
7              Thread.sleep(1000);
8          } catch (InterruptedException e) {
9
10             e.printStackTrace();
11         }
12     }
13 }
```

Secure Code

- In the below code, run() method appropriately **catches** the **InterruptedException**
- It restores the status of the current thread by promptly calling **interrupt()** method

Demo.java

```
1
2  public class Demo implements Runnable {
3
4      @Override
5      public void run() {
6          try {
7              Thread.sleep(1000);
8          } catch (InterruptedException e) {
9
10             Thread.currentThread().interrupt();
11         }
12     }
13 }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Erroneous Exceptional Behaviors: Suppressing or Ignoring Checked Exceptions (Cont'd)

Vulnerable Code

Demo.java

```
1
2  public class Demo implements Runnable {
3
4      @Override
5      public void run() {
6          try {
7              Thread.sleep(1000);
8          } catch (InterruptedException e) {
9
10             e.printStackTrace();
11         }
12     }
13 }
```

- The code prints the exception's **stack trace**
- Exception's stack trace is used for **debugging purposes** but it may also results in suppressing the exception
- Printing the stack trace can provide information to attacker about the **structure** and **state** of the **process**
- This code snippet does not evaluate the **expressions** or **statements** that occur after the **try block** throws exceptions

Secure Code

- The secure code uses **FileNotFoundException**
- It requests the user to specify the **desired file name**

Demo.java

```
9
10  public static void main(String args[]) throws FileNotFoundException
11  {
12      String fileName = "temp.txt";
13      String line = null;
14          FileReader fileReader = new FileReader(fileName);
15              BufferedReader bufferedReader =  new BufferedReader(file
16
17              try {
18                  while((line = bufferedReader.readLine()) != null) {
19                      System.out.println(line);
20                  }
21              } catch (IOException e) {
22
23                  e.printStackTrace();
24              }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Erroneous Exceptional Behaviors: Disclosing Sensitive Information

- Throwing exceptions without filtering sensitive information may result in **information leaks**
- It may provide attackers with information that can be used to perform **further exploits**

Secure Code

- The code implements the policy that only files in the **c:\homepath** can be accessed by the user
- **File.getCanonicalFile()** method is also used to **canonicalize** the file subsequent path name

Vulnerable Code

- The example code provides a user with **contents** and **layout** of the **file system**

*Demo.java

```
5   public static void main(String args[]) throws FileNotFoundException
6   {
7           try {
8           FileInputStream fInSt = new FileInputStream(System.getenv("APPDATA") +args[0]);
9           } catch (FileNotFoundException x)
10          {
11          System.out.println("Invalid file");
12          return;
13          }
14  }
```

Demo.java

```
10
11
12  public static void main(String args[]) throws FileNotFoundException
13  {
14      File f1 = null;
15  try {
16  f1 = new File(System.getenv("APPDATA") +
17  args[0]).getCanonicalFile();
18          if (!f1.getPath().startsWith("c:\\homepath"))
19              {
                tem.out.println("Invalid File");

            ption e)

            .out.println("Invalid file");


            utStream fInSt = new FileInputStream(f1);
            (FileNotFoundException x)

            .out.println("Invalid file");
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Erroneous Exceptional Behaviors: Disclosing Sensitive Information (Cont'd)

Vulnerable Code

Demo.java

```
8   public static void main(String args[]) throws IOException
9   {
10      try {
11          FileInputStream fInSt = new FileInputStream(System.getenv("APPDATA") +args[0]);
12          } catch (FileNotFoundException x)
13          {
14              Logger logger = Logger.getAnonymousLogger();
15
16              logger.log(Level.SEVERE, "thrown Exception", x);
17
18              throw new IOException("Unable to retrive file",x);
```

- The example code throws a general exception after a logged exception **leaks** the **file system layout information** to the attacker

- The code implements the policy that only files in the **c:\homepath\file1** and **c:\homepath\file2** can be accessed by the user

Secure Code

porter.java

```
.Logger;
6  public class Demo {
7
8      public static void main(String args[]) throws IOException
9      {try {
10         FileInputStream fInSt =null;
11         switch(Integer.valueOf(args[0])) {
12         case 1:
13             fInSt=new FileInputStream("c:\\homepath\\file1");
14         case 2:
15             fInSt=new FileInputStream("c:\\homepath\\file2");
16             default:
17                 System.out.println("Invalid Option");
18                 break;
19         }
20         } catch (Throwable t) {
21         MyExceptionReporter.report(t);
22         }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Erroneous Exceptional Behaviors: Logging Sensitive Data

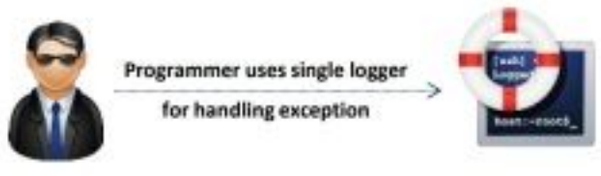
- Throwing exceptions while logging a process can sometimes **disrupt** the **process**, and failure to throw exceptions can create **security vulnerabilities**
- Programmers should ensure that the logging process should not be disrupted by **throwing incorrect exceptions**

Vulnerable Code

- The code snippet below writes **SecurityException** to the standard error stream that can be inadequate for the logging process
- Standard error stream can be **exhausted or closed** and may further prevent exceptions from recording
- Error streams used for certain **security-related exceptions** or errors may not provide adequate security
- If any input/output errors occur while writing security exceptions, the catch block may throw an **IOException** leaking security information to the attacker
- Using **System.out.print*()**, Console.printf() or Throwable.printStackTrace() to print may leak security information

Secure Code

- The code snippet below uses **java.util.logging.Logger** (default logging API in JDK 1.4)
- It attempts to simplify the entire logging process by using only one **logger**



Demo.java | MyExceptionReporter.java

```
10      try {
11          FileInputStream fInSt = new FileInputStr
12          } catch (SecurityException x)
13          {
14              System.err.println(x);
15          }
```

Demo.java | MyExceptionReporter.java

```
10      try {
11          FileInputStream fInSt = new FileInputStream(System.getenv("APPDATA") +args[0]);
12          } catch (SecurityException x)
13          {
14              Logger logger =Logger.getAnonymousLogger();
15
16              logger.log(Level.SEVERE, "thrown Exception", x);
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Erroneous Exceptional Behaviors: Restoring Objects to Prior State, if a Method Fails

- Objects (general and security-related) should be maintained in a **consistent manner** even if exceptions arise

Objects consistent methods include:

1. Using **rollback** during failure events
2. Rearrange **exceptional condition mechanism** so that the exception condition executes before the modification of the object
3. Any **changes** to the **object** should be avoided
4. Input validation
5. Operations that require execution should be implemented on a temporary **copy** of the **object** rather than the original object and committing those **changes** to the **original object**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Erroneous Exceptional Behaviors: Restoring Objects to Prior State, if a Method Fails (Cont'd)

Vulnerable Code

- The code example uses dimension class with three **internal attributes** length, width and height of a rectangular box
- The **getVolumePackage()** method is used for returning the total volume of the box after accounting for packaging material
- getVolumePackage() method returns volume
- Input validation rejects **negative values**
- Input values cannot be more than 10, if an input is more than 20 then **illegalArgumentException** is **thrown**

Demo.java | MyExceptionReporter.java

```
1  public class Demo {
2
3      private int length;
4      private int width;
5      private int height;
6      static public final int PADDING=2;
7      static public final int MAX_DIMENSION = 10;
8
9      public  Demo(int length, int width, int height) {
10         this.length=length;
11         this.width=width;
12         this.height=height;
13     }
14
15
16     protected int getVolumePackage(int weight) {
17         length+= PADDING;
18         width+= PADDING;
19         height += PADDING;
20         try {
21         if(length <= PADDING || width <= PADDING || height <= PADDING ||
22         length > MAX_DIMENSION + PADDING || width > MAX_DIMENSION + PADDING ||
23         height > MAX_DIMENSION + PADDING || weight <= 0 || weight > 20)
24         {
25             throw new IllegalArgumentException();
26         }
27
28         int volume = length * width * height;
29         length -= PADDING; width -= PADDING; height -= PADDING;
30         return volume;
31         } catch (Throwable t)
32         {
33             MyExceptionReporter mer = new MyExceptionReporter();
34             mer.report(t);
35         }
36         return -1;
37
38 }
39     public static void main(String[] args) {
40     Demo d = new Demo(10, 10, 10);
41     System.out.println(d.getVolumePackage(21)); // Prints -1 (error)
42     System.out.println(d.getVolumePackage(19)); // Prints 2744 instead of 1728
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Erroneous Exceptional Behaviors: Restoring Objects to Prior State, if a Method Fails (Cont'd)

Secure Code

Example with **Input Validation**

- The secure code performs **input validation** before modifying the object state
- The try block in the code considers only those statements that **throw exceptions** removing others outside the try block

```
Demo.java  MyExceptionReporter.java
15
16    protected int getVolumePackage(int weight) {
17        length+= PADDING;
18        width+= PADDING;
19        height += PADDING;
20        try {
21        if(length <= PADDING || width <= PADDING || height <= PADDING ||
22        length > MAX_DIMENSION + PADDING || width > MAX_DIMENSION + PADDING ||
23        height > MAX_DIMENSION + PADDING || weight <= 0 || weight > 20)
24        {
25            throw new IllegalArgumentException();
26        }
27
28        int volume = length * width * height;
29        length -= PADDING; width -= PADDING; height -= PADDING;
30        return volume;
```

Secure Code

Example with **Rollback**

- The code example uses a catch block replacing the **getVolumePackage()** method
- This code aims at restoring prior **object state** if any exceptional event occurs

```
Demo.java  MyExceptionReporter.java
27
28        int volume = length * width * height;
29        length -= PADDING; width -= PADDING; height -= PADDING;
30        return volume;
31        } catch (Throwable t)
32        {
33            MyExceptionReporter mer = new MyExceptionReporter();
34            mer.report(t);
35        }
36        return -1;
37    }
38
```



Erroneous Exceptional Behaviors: Avoid using Statements that Suppress Exceptions

- Use of **return**, **break**, **continue** or **throw statements** is restricted in a finally block
- When a program executes a try or catch block , the finally block in the try block is executed, sending the program to **normal completion**
- Statements used in the finally block may cause the try block to **complete abruptly** and suppress any exceptions thrown from the try or catch blocks

Vulnerable Code

```
Demo.java  MyExceptionReporter.java
3  public static boolean PerformLogic()
4  {
5    try {
6
7        throw new IllegalStateException();
8        }
9    finally {
10       System.out.println("finally Logic Performed..");
11   return true;
12   }
13 }
```

- In the code, **return** statement used in the finally block completes abruptly

Secure Code

```
Demo.java  MyExceptionReporter.java
1  public class Demo {
2  public static boolean PerformLogic()
3  {
4      try {
5          throw new IllegalStateException();
6
7          finally {
8              System.out.println("finally Logic Performed..");
9          }
10
11     //return true;
12       //return statements must go here if any;
13       //applicable only when exception is thrown conditionally
14 }
15 }
```

- In the below code, return statement is removed from the finally block

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Erroneous Exceptional Behaviors: Prevent Access to Untrusted Code that Terminates JVM

- **System.exit()** when invoked can even terminate the Java Virtual Machine (JVM) resulting in the termination of currently running programs as well as threads that might even lead to **Denial-of-Service (DoS) attacks**

Vulnerable Code

Demo.java | MyExceptionReporter.java

```
1  public class Demo {
2
3      public static void main(String args[])
4      {
5          //.................
6          //........
7
8          System.exit(1);// Exit Abruptly
9          System.out.println("This will never Execute");
10
11     }
12 }
13
```

- The above code shows how the JVM is shutdown and the running process is terminated by using **System.exit()**
- The code also does not have **security manager** to check if the caller is valid or not to invoke the **System.exit()**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Erroneous Exceptional Behaviors: Prevent Access to Untrusted Code that Terminates JVM (Cont'd)

Secure Code

- The code contains security manager "**passwordSecurityManager**" that overrides the **checkExit()** method of the **SecurityManager** class
- This installed security manager tracks if **exit** is permitted or not
- It also does necessary **cleanup action** as well as **logging** the **exception**

PasswordSecurityManager.java

```
1
2  public class PasswordSecurityManager extends SecurityManager {
3      private boolean Flag;
4      public PasswordSecurityManager(){
5          super();
6          Flag = false;
7      }
8      public boolean ExitAllowed(){
9          return Flag;
10
11     }
12     @Override
13     public void checkExit(int status) {
14         if (!ExitAllowed()) {
15             throw new SecurityException();
16         }
17         super.checkExit(status);
18     }
19     public void AllowedExit(boolean b) {
20         Flag = b;
21     }
22 }
23 class TerminateExit{
24     public static void main(String[] args) {
25         PasswordSecurityManager PSM=PasswordSecurityManager();
26         System.setSecurityManager(PSM);
27         try {
28             //..
29             System.exit(1);
30         } catch (Throwable t) {
31             if(t instanceof SecurityException)
32             {
33                 System.out.println("Interupted System.exit(1)");
34                 //Log exception
35             }else {
36                 //handover to Exception Handler
37             }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Erroneous Exceptional Behaviors: Never Catch java.lang.NullPointerException

- **java.lang.NullPointerException** and **RuntimeException** should not be caught by the programs
- A NullPointerException indicates about the **existence of null pointer** dereference by throwing an exception at runtime
- **Bugs** in the program are indicated by the Runtime exceptions need to be fixed by the programmer

Vulnerable Code

- The following code gives an example based on `isName()` method
- In this method a String argument is taken and if the given string is a valid name then it returns true
- When this method fails to check whether the given string is null or not and instead catches `NullPointerException`, then it returns `false`

Secure Code

- The following code checks if the given string is null or not instead of catching the `NullPointerException`

Demo.java

```
1  public class Demo {
2
3          boolean isName(String str) {
4              try {
5                  String sNames[] = str.split(" ");
6                  if(sNames.length != 2) {
7                      return false;
8                  }
9                  return (isCapitalized(sNames[0]) && isCapitalized(sNames[1]));
10             } catch (NullPointerException e) {
11                 return false;
12             }
13         }
```

```
ss Demo {

olean isName(String str) {
  try {
    if (str == null) {
      return false;
    }
    String sNames[] = str.split(" ");
    if(sNames.length != 2) {
      return false;
    }
    return (isCapitalized(sNames[0]) && isCapitalized(sNames[1]));
  } catch (NullPointerException e) {
    return false;
  }
15 }
```

Erroneous Exceptional Behaviors: Never Allow Methods to Throw RuntimeException, Exception, or Throwable

- **RuntimeException**, **Exception**, or **Throwable** should not be thrown or else it may lead to various errors

Vulnerable Code

- The following code uses `toUpperCase()` method
- This method accepts a string and **returns true** if the string is a capital letter followed by lowercase letters
- When a null string argument is passed this method throws a `RuntimeException`

Secure Code

- The following code throws a `NullPointerException` in order to denote exceptional condition

test.java

```
1  package Test;
2
3  public class test {
4  boolean isName(String str)
5  {
6    if(str==null)
7    {
8      throw new RuntimeException("Null String");
9
10   }
11   if(str=="")
12   {return true;}
13   String begin=str.substring(0, 1);
14   String next=str.substring(1);
15   next.equals(next.toLowerCase());
16   return begin.equals(begin.toUpperCase());
17 }
```

Demo.java

```
1  public class Demo {
2
3          boolean isName(String str) {
4
5                  if (str == null) {
6                      throw new NullPointerException();
7                  }
8                  if(str=="")
9                  {
10                     return true;
11                 }
12                 String begin=str.substring(0, 1);
13                 String next =str.substring(1);
14                 return (begin.equals(begin.toUpperCase()));
15
16         }
```


Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Erroneous Exceptional Behaviors: Never Throw Undeclared Checked Exceptions

1. In Java, there are a few techniques that permit **throwing** the undeclared checked exceptions at **runtime**
2. These techniques do not allow the usage of **throws clause** by weakening the ability of **caller methods**
3. These techniques should never be used to throw undeclared checked exceptions

Vulnerable Code

- The given vulnerable code **throws undeclared checked exceptions**
- This code uses the **undelcaredThrow()** method that takes a **Throwable** argument and then invokes a function that throws the argument without declaration

Demo.java

```
1  public class Demo {
2            private static Throwable t;
3            private Demo() throws Throwable {
4              throw t; }
5          public static synchronized void undeclaredThrow(Throwable
6              // These exceptions must not be passed
7          if (t instanceof IllegalAccessException ||
8                  t instanceof InstantiationException) {
9                // Unchecked, declaration not required
10               throw new IllegalArgumentException(); }
11             Demo.t = t;
12             try {
13         //Throwable argument passed above is thrown in next line
14         // even though it throws clause of class.newInstance fail
15               // to declare that this may happen
16               Demo.class.newInstance();
17         } catch (InstantiationException ie) { /* unreachable */
18         } catch (IllegalAccessException ie) { /* unreachable */
19             } finally { // Avoid memory leak
20               Demo.t = null;
21             } }
22
23
24         public static class UndeclaredException {
25               public static void main(String[] args) {
26                 // No declared checked exceptions
27                 Demo.undeclaredThrow(
28                     new Exception("Some checked exception"));
29         }
30         }
31 }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Erroneous Exceptional Behaviors: Never Throw Undeclared Checked Exceptions (Cont'd)

Secure Code

- This solution code uses **java.lang.reflect.Constructor.newInstance()** instead of **Class.newInstance()**
- The **Constructor.newInstance()** process shawls any exceptions thrown from within the constructor into a checked exception known as **Invocation Target Exception**

Demo.java

```
37
38         public static synchronized void undeclaredThrow1(Throwable t) throws NoSuchMethodException, SecurityException {
39               // These exceptions should not be passed
40               if (t instanceof IllegalAccessException ||
41                   t instanceof InstantiationException) {
42                 // Unchecked, no declaration required
43                 throw new IllegalArgumentException(); }
44               Demo.t = t;
45               try {
46                 Constructor<Demo> constr =Demo.class.getConstructor(new Class<?>[0]);
47                 constr.newInstance();
48               } catch (InstantiationException ie) { /* unreachable */
49               } catch (IllegalAccessException ie) { /* unreachable */
50               } catch (InvocationTargetException ie) {
51                 System.out.println("Exception thrown: "+ ie.getCause().toString());
52               } finally { // Avoid memory leak
53                 Demo.t = null;
54               } }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Erroneous Exceptional Behaviors: Never Let Checked Exceptions Escape from Finally Block

- Methods invoked from within a **finally** block throw an **exception**
- If such exceptions are not caught and handled then it results in termination of entire **try block**

Vulnerable Code

Invoke.java

```
1  import java.io.BufferedReader;
2  import java.io.FileReader;
3  import java.io.IOException;
4
5  public class Invoke {
6  public static void doInvoke(String file) {
7  // ... code to check or set character encoding ...
8      try {
9        BufferedReader rdr =
10           new BufferedReader(new FileReader(file));
11       try {
12         // Do operations
13       } finally {
14         rdr.close();
15         // ... Other cleanup code ...
16       }
17     } catch (IOException ix) {
18       // Forward to handler
19     } } }
```

- This vulnerable code has a finally block which **closes reader object**
- It is wrongly assumed by the programmer that the **statements** of the **finally block** do not throw exceptions
- This results in the **failure** of handling **exceptions**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Erroneous Exceptional Behaviors: Never Let Checked Exceptions Escape from Finally Block (Cont'd)

Secure Code

Invoke.java

```
1  import java.io.BufferedReader;
2  import java.io.FileReader;
3  import java.io.IOException;
4
5  public class Invoke {
6      public static void doInvoke(String file) {
7          // ... code to check or set character encoding ...
8          try {
9              BufferedReader rdr = new BufferedReader(new FileReader(file));
10             try {
11                 // Do operations
12             } finally {
13                 try {
14                     rdr.close();
15                 } catch (IOException ie) {
16                     // Forward to handler }
17                     // .........Other clean-up code ...
18                 }
19             }
20         } catch (Exception e) {
21             // TODO: handle exception
```

- The code enfolds the `close()` method request in a **try-catch block** in the finally block
- Therefore, the potential `IOException` is held without allowing it to circulate anymore

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Dos and Don'ts in Error Handling

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Dos and Don'ts in Exception Handling

- The **Exception class** should not be used directly in the code
- It should be classified into **RuntimeExceptions** and **Errors**

Don'ts

Invoke.java

```
1
2  public class Invoke {
3
4      public static void main(String args[])
5      {
6          try {
7              //Statements
8          } catch (Exception e) {
9              // TODO: handle exception
10         }
11     }
```

Dos

Invoke.java

```
11
12             //...statements
13
14
15
16         } catch (Exception ex) {
17             if(ex instanceof RuntimeException) throw (RuntimeException)ex;
18             if(ex instanceof org.apache.tomcat.jni.Error)throw (Error)ex;
19
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Dos and Don'ts in Exception Handling (Cont'd)

- The **printStackTrace()** function should not be used in the code as it reveals all the exception information to the user
- An attacker can **misuse** the information to perform attack on the application
- The exception thrown by an application should be stored in the **logger file** using a logger application

Don'ts

Invoke.java

```
14
15
16          } catch (Exception ex) {
17              ex.printStackTrace();
18
19          }
20      }
21
22  }
```

Dos

HomeControll... | Userform.java | MyException...

```
11      {
12          try {
13
14          }catch(Exception ex){
15
16              Logger logger=Logger.getLogger("Stack-trace-logger");
17              logger.log(level, "logger", ex);
18
19          }
20          }
21
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Dos and Don'ts in Exception Handling (Cont'd)

- The vulnerable code throws an **exception handler** function with the same name from all user functions (isA, isB and isC)
- This is not a good practice, because when an the **error** is **encountered**, it becomes difficult for the developer to identify the exact function that has thrown the error
- It is a good practice to **code separate error functions** for every user function

Don'ts

MyDomainEg.java

```
1
2  public interface MyDomainEg {
3
4          void isA()
5          throws
6           MyDomainException;
7          void isB()
8           throws
9          MyDomainException;
10         void isC()
11          throws MyDomainException;
12
```

Dos

MyDomainEg.java

```
1  public interface MyDomainEg {
2  void isA() throws SomeMyDomainException,OtherMyDomainException;
3
4  void isB()throws SomeMyDomainException,OtherMyDomainException;
5  }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Dos and Don'ts in Exception Handling (Cont'd)

- Clean up code such as releasing resources, closing input I/O streams, and deleting files should be done in the **finally** block

Don'ts

DemoForException.java

```
1  import java.io.BufferedReader;
2  import java.io.FileNotFoundException;
3  import java.io.FileReader;
4  import java.io.IOException;
5
6  public class DemoForException {
7
8      public void Readfile()
9      {
10     FileReader fr = null;
11     String file=null;
12     try {
13         fr = new FileReader(file);
14         BufferedReader br = new BufferedReader(fr);
15         String line = null;
16         fr.close();
17     } catch (FileNotFoundException e) {
18
19         e.printStackTrace();
20     }
```

- Clean up code should not be kept in **try** or **catch** block

Dos

DemoForException.java

```
10      FileReader fr = null;
11      String file=null;
12      try {
13          fr = new FileReader(file);
14          BufferedReader br = new BufferedReader(fr);
15          String line = null;
16
17      } catch (FileNotFoundException e) {
18
19          e.printStackTrace();
20      } finally {
21          if (fr != null) {
22              try {
23                  fr.close();
24              } catch (IOException e) {
25
26                  e.printStackTrace();
27              }
28          }
```

- Clean up code should be kept only in **finally** block

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Dos and Don'ts in Exception Handling (Cont'd)

- Exceptions should be logged properly using a **logger file**

Don'ts

Invoke.java

```
1
2  public class Invoke {
3
4      public static void Test()
5      {
6          try {
7              int i=5;
8              i=i/0;
9
10         } catch (Exception ex) {
11             ex.printStackTrace();
12         }
13     }
```

Dos

Invoke.java

```
1  import java.util.logging.Level;
2  import java.util.logging.Logger;
3
4  public class Invoke {
5
6      public static void Test()
7      {
8          try {
9              int i=5;
10             i=i/0;
11
12         } catch (Exception ex) {
13             Logger logger=Logger.getLogger("Stack-Trace-Logger");
14             logger.log(Level.SEVERE, "thrown Exception", ex);
15
16             throw new ArithmeticException("Devide By zero");
17
18
19         }
20     }
21     public static void main(String[] args) {
22         Test();
23     }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Avoid using Log Error and Throw Exception at Same Time

- Throwing and logging exception at same time creates multiple **logs**

The following example shows the improper and proper approach

Improper Approach

Invoke.java

```
1  import org.apache.log4j.Logger;
2  import org.apache.log4j.LogManager;
3
4  public class Invoke {
5
6      public static void Test()
7      {
8          final Logger logger = Logger.getLogger(Invoke.class);
9          try {
10             int i=5;
11             i=i/0;
12
13         } catch (Exception ex) {
14             logger.error("The Error is", ex);
15
16         }
17     }
```

catch **Exception** class and log an Exception

Proper Approach

Invoke.java

```
import org.apache.log4j.Logger;
import org.apache.log4j.LogManager;

public class Invoke {

    public static void Test()
    {
        final Logger logger = Logger.getLogger(Invoke.class);
        try {
            int i=5;
            i=i/0;

        } catch (ArithmeticException ex) {
            System.out.println("The Error is.."+ex.getMessage());

        }
    }
```

Handled only **Exception** instead log an error

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Spring MVC Error Handling

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Spring MVC Error Handling

The following are different exception handling **approaches** in Java Spring MVC Framework

Controller Level Exception Handler

- Controller exception can be handled by:
  - Adding **@ExceptionHandler** annotation
  - Implementing **HandlerExceptionResolver**

Global Exception Handler

- Adding **@ControllerAdvice** annotation to define global exceptions
- Using **HandlerExceptionResolver** interface for creating global exception handlers

Custom Errors Mapping

- Adding **@ResponseStatus** annotation to map custom errors to exception status code

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Handling Controller Exceptions with @ExceptionHandler Annotation

- **To Implement @ExceptionHandler annotation**
  - Add **@ExceptionHandler** annotation to the method
  - Include a list of exceptions to be handled by the method
  - When handling multiple exceptions, **specific exceptions** are thrown ignoring **general exceptions**
  - Return error response as string, Modelview object, ResponseEntity or @Responsebody

Vulnerable Code: Unhandled Exception

ErrorHandling.java

```
16  @RequestMapping("/addNewUser")
17  public String newUser(Model model) throws Unauthor
18  {
19  model.addAttribute("username", new User());
20  List<String> prefContactMethods=new ArrayList<Stri
21  prefContactMethods.add("E-mailaddress");
22  prefContactMethods.add("CallNo");
23  model.addAttribute("prefContctName", prefContctMet
24  List<String> sexType=new ArrayList<String>();
25  sexType.add("Male");
26  sexType.add("Female");
27  model.addAttribute("sexcat", sexcat);
28  if(true)
29  throw new UnauthorizedAccessException();
30  return "newUser";
31  }
```

Secure Code: Handling Exception using @ExceptionHandler Annotation

AdminCourseController.java | RestController.java

```
1  package com.ECouncil.CASE;
2
3  import org.springframework.http.HttpStatus;
4  import org.springframework.http.ResponseEntity;
5  import org.springframework.web.bind.annotation.ExceptionHandler;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RequestMethod;
8
9  @RestController
10 @RequestMapping("/admincourses")
11 public class AdminCourseController {
12     @RequestMapping(method = RequestMethod.GET)
13     public ResponseEntity<Object> get(){ throw new RuntimeException("courses not yet supported");
14     }
15
16     @ExceptionHandler(RuntimeException.class)
17     public ResponseEntity<Error> handle(RuntimeException ex){
18     System.out.println("controller local exception handling @ExceptionHandler");
19     Error error = new Error(ex.getMessage());
20     return new ResponseEntity<Error>(error, HttpStatus.INTERNAL_SERVER_ERROR);
21     }
22 }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Handling Controller Exceptions with HandlerExceptionResolver

- **To implementing HandlerExceptionResolver Interface**
  - Override the resolveException() method
  - It will handle all the exceptions of the controller
  - Use **instanceof** to get the exception type

Example: Controller Exception Handling using HandlerExceptionResolver

ECEmployee.java

```
2
3  import javax.servlet.http.HttpServletRequest;
4  import javax.servlet.http.HttpServletResponse;
5
6  import org.springframework.http.HttpStatus;
7  import org.springframework.web.servlet.HandlerExceptionResolver;
8  import org.springframework.web.servlet.ModelAndView;
9  import org.springframework.web.servlet.view.json.MappingJacksonJsonView;
10 public class ECEmployee implements HandlerExceptionResolver {
11     @Override
12     public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler,
13             Exception ex) {
14         System.out.println("exception handling HandlerExceptionResolver"); response.reset();
15         response.setCharacterEncoding("UTF-8");
16         response.setContentType("text/json");
17         ModelAndView model=new ModelAndView(new MappingJacksonJsonView());
18         if(ex instanceof RuntimeException)
19         {
20             response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
21             model.addObject("errorcode", HttpStatus.INTERNAL_SERVER_ERROR.value());
22             model.addObject("errormessage", ex.getMessage());
23                     }
24         else
25         {   response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
26              model.addObject("errcode", HttpStatus.INTERNAL_SERVER_ERROR.value()); model.addObject("errmessage", ex.getMessage());
27
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Spring MVC: Global Exception Handling

- For global exception handling use **@ControllerAdvice** annotation along with **@ExceptionHandler** method annotation for any class

Example: Global Exception Handling using @ControllerAdvice

MyProjectExceptionHandler.java | ControllerAdvice.java

```
1  package com.ECouncil.CASE;
2
3  import org.springframework.http.HttpStatus;
4  import org.springframework.http.ResponseEntity;
5  import org.springframework.web.bind.annotation.ExceptionHandler;
6
7  @ControllerAdvice
8  public class MyProjectExceptionHandler extends ResponseEntityExceptionHandler {
9
10
11     @ExceptionHandler(RuntimeException.class)
12     public ResponseEntity<Error> handle(RuntimeException ex){
13         Error error = new Error(ex.getMessage());
14         return new ResponseEntity<Error>(error, HttpStatus.INTERNAL_SERVER_ERROR);
15     }
16
17 }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Global Exception Handling: HandlerExceptionResolver

- Create a custom global error handler by implementing **HandlerExceptionResolver** interface
- It enables mapping exceptions to a view name

Example: MappingExceptionResolver - with Java Configuration

MyProjectExceptionHandler.java

```
@Bean
public ExampleMappingExceptionResolver exceptionResolver()
{
    ExampleMappingExceptionResolver resolver = new ExampleMappingExceptionResolver();
    Properties exceptions = new Properties();
    exceptions.put(ArithmeticException.class, "error");
    resolver.setExceptionMappings(exceptions);
    return resolver;

}
}
```

Example: MappingExceptionResolver - with XML Configuration

Beans.xml

```
<bean class="org.springframework.web.servlet.handler.ExampleMappingExceptionResolver">
<property name="exceptionMappings"> <props>
<prop key="java.lang.ArithmeticException">error</prop> </props>
 </property>
  </bean>
</beans>
```

Global Exception Handling: HandlerExceptionResolver (Cont'd)

CustomError.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java"%>
<html>
<head>
<title>Exception Handling</title>
</head>
<body>
    <h1>Sorry For </h1>
    ${exception.message}
</body>
</html>
```

Design | Preview

Sample Custom Error Page

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Mapping Custom Exceptions to Statuscode with @ResponseStatus

- Adding **@ResponseStatus** annotation enables mapping custom exceptions with default spring status codes
- **ResponseStatusExceptionResolver** automatically handles all thrown exceptions

Default Exception Handling Status Codes

Status Code	Exception
404	Not Found–>NoSuchRequestHandlingMethodException etc.
400	Bad Request–>BindException, HttpMessageNotReadableException etc.
406	Not Acceptable–>HttpMediaTypeNotAcceptableException
415	Unsuppported Media Type–>HttpMediaTypeNotSupportedException
405	Method Not Allowed–>HttpRequestMethodNotSupportedException
500	Internal Server Error–>HttpMessageNotWritableException

Example: Status Code Mapping to Custom Exception

Beans.xml | CustomError.jsp | ControllerA... | EmployeeNotA...

```
1  package com.ECouncil.CASE;
2
3  import org.springframework.http.HttpStatus;
4  import org.springframework.web.bind.annotation.ResponseStatus;
5
6  @ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "")
7  public class EmployeeNotAvailableException extends RuntimeException
8  {
9      private static final long serialVersionUID = 1L;
10     EmployeeNotAvailableException()
11     {
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Configure Custom Error Page in Spring MVC

- By default, Spring MVC will raise an exception while trying to access unauthorized resources. Such exceptions should be handled correctly by customizing **error page configuration** in Spring MVC
- It is also possible to configure custom error pages using Java configuration with the help of **@EnableWebSecurity** annotation and **WebSecurityConfigurerAdapter**

XML configuration

web.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <http>
4     <intercept-url pattern="/admin/*" access="hasAnyRole('ROLE_ADMIN')"/>
5         <access-denied-handler error-page="/my-error-page" />
6 </http>
```

Design | Source

Note: If an unauthorized user try to access a page then response will be redirected to "/my-error-page"

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Exception Handling in Struts 2

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Exception Handling: Struts 2

Global Exception Handling

- To implement **global exception handling** include the given struts.xml file. This will map exceptions of all action classes

Global Exception Configuration

struts.xml

```
33 <global-results>
34 <result name="globalerror">/Errorpage.jsp</result>
35 </global-results>
36 <global-exception-mappings>
37     <exception-mapping exception="java.lang.Exception" result="globalerror"/>
38 </global-exception-mappings>
39
40 </package>
```

Action Level Exception handing

- To implement **action level exception handling** add <exception-mapping> elements within the <action> element of the **struts.xml** file

Action Level Exception Configuration

struts.xml

```
23 </global-exception-mappings>
24
25     <action name="EmployeeAction" class="Com.ECProject.Info.EmployeeAction">
26     <exception-mapping result="databaseError" exception="java.sql.SQLException"></exception-mapping>
27     <result name="success">/DbConnection.jsp</result>
28     <result name="databaseError">/DbConnection.jsp</result>
29     </action>
```

Will redirect to DBError.jsp when action "connectDB" thows any exception

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

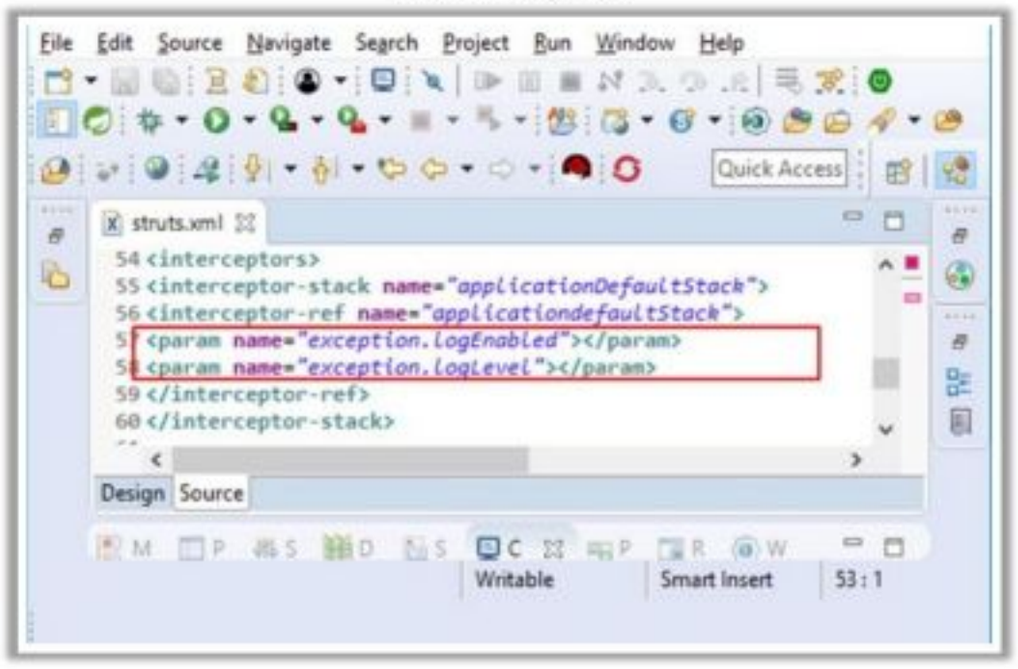
@ErrorCatchers

Exception Handling: Struts 2 (Cont'd)

Logging Exception

- Log exceptions to page and/or log files
- To enable **logging** declare an interceptor as shown:

Enable Logging



Note: The code snippet should be included in struts.xml

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Best Practices for Error Handling

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Best Practices for Handling Exceptions in Java

1. Throw exceptions that are relevant to the interface

- A function should throw exceptions that are relevant to its **interface**

  Example: `java.io.FileInputStream.FileInputStream(String name)` throws `FileNotFoundException`
- The above code is used for opening a file on disk for reading bytes
- The given constructor may throw a `FileNotFoundException` and not `IndexOutOfBoundsException` that is completely irrelevant to the code
- Developers should make sure to throw **exceptions** that are **relevant** to the interface

2. The reason for exception should be specified

- The exceptions should always **specify what caused** them to occur
- Exceptions can be handled in either of the following two ways:
  - Generating a **separate exception** for each and every event
  - Creating a **generic exception** and mentioning what caused that exception
- The best practice is to use an exception class when handling specifically with **separate code**
- Use a single generic exception class for handling exceptions in a **default way**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Best Practices for Handling Exceptions in Java (Cont'd)

3. Encapsulation

- In a scenario where a function throws an exception that is received from another function, it should be encapsulated in a locally generated exception class
- In the sample code, `FileNotFoundException` is converted into `ActionException`
- When the code encounters the `FileNotFoundException` inside the `try` block, the `catch` clause throws an `ActionException`
- The execution thread is suspended and the exception is reported. The function `getCause()` can be made use of to find the root exception cause

Example Code

General.java

```
5
6  public class General {
7      public static FileOutputStream main(String sname) thr
8      {
9
10         try {
11             String fName = sname;
12            return new  FileOutputStream(fName);
13            } catch (FileNotFoundException fe) {
14             throw new ActionException(fe);
15            }
16
17     }
18 }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Best Practices for Handling Exceptions in Java (Cont'd)

4. Balance the catch blocks

- There are two approaches that can be used to balance the `catch` blocks:

The first approach is to use a **single `catch` block** for all the events.

The second approach is to use **multiple `catch` blocks** .i.e., each exception will have a **separate `catch` block**

For example: Many catch blocks one for each Exception

General.java

```
1  package com.ec.web;
2
3  public class General {
4      public static void main(String args[]) throws Command
5      try {
6
7      } catch (Throwable t) {
8          throw new CommandExecutorException(t);
9      }
10
11     }
12 }
```

General.java

```
4      try {
5
6          System.out.println("Here Statements");
7      }
8      catch (ArrayIndexOutOfBoundsException  ex) {
9
10     }
11     catch (ArithmeticException a) {
12
13     }catch(RuntimeException x) {}
14
15     }
```

- This approach is not a good practice as it does not give control over the exceptions being **caught**
- Developers should follow this approach as it is easy to execute various operations depending on the exceptions **thrown**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Best Practices for Handling Exceptions in Java (Cont'd)

5. Create custom error pages

- A generic error page should be used for **all exceptions**
- This prevents an **attacker** from retrieving internal response to errors

This can be done using the following ways:

1. **Declarative Exception Handling using Struts**
2. **Using Java Servlets**

struts-config.xml

```
17
18 </action-mappings>
19 <global-exceptions>
20 <exception
21  type="ECProject.Info.EmployeeNotFoundException"
22  path="/Sorry.jsp"
23  key="EC.proj.SorryAct">
24 </exception>
25 </global-exceptions>
26
27 </struts-config>
```

Design | Source

- The code should be included in struts-config.xml

web.xml

```
3 <error-page>
4 <exception-type>EmployeeNotFoundException</exception-type>
5 <location>Sorry.jsp</location>
6 </error-page>
```

Design | Source

- The code should be implemented in web.xml

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Best Practices for Handling Exceptions in Java (Cont'd)

6. Exception names should be meaningful

- The exception names must be **meaningful** and should **express** the **action** that caused them
- Programmers should make sure to use **specific exceptions** and not generic exceptions
- Example: **java.io.FileNotFoundException** can be used as it implies that file is not found
- **java.lang.Exception** should not be used as it is very **generic**

7. Throw early and catch late

- This principle should be followed by all **programmers**
- The exception should be **thrown early** and **caught late**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Introduction to Logging

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Logging in Java

- In Java, logging frameworks are provided by the package **java.util.logging** and used to write log messages on to a central location
- The log messages can be **error messages**, **informational messages** or **configuration messages**

Logging is divided into three parts:

Logger	Receives the message that is then passed to the **logging framework** The framework then calls the **formatter** including the message
Formatter	Formats the **message** for output The framework then **passes** the **output** to the handler
Handler (Appender)	The appropriate **handler dispositions** the output Examples of Java Logging Frameworks: Apache Commons Logging, Java Logging API, Log4J, SLF4J

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Example for Logging Exceptions

- **The code below gives an example of logging an exception to a log file**

Invoke.java

```
7     Logger.getLogger(SimpleClass.class.getName());
8   public void writeLog() {
9     logger.info("Enable logging...");
10    try {
11       throw new Exception("Simulating an exception");
12    } catch (Exception e){
13       logger.log(null, e.getMessage());
14    }
15  }
16   public static void main(String[] args) {
17
18    SimpleClass simpleObj = new SimpleClass();
19    simpleObj.writeLog();
20    logger.info("Done...");
21  }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Logging Levels

Logging plays a major role in web applications as it is the only way to track the server's activities. Logging levels based on Java log4j are as follows:

DEBUG
- The lowest restricted Java logging level
- The messages logged in this level are used to debug the application
- It is used only in the development and testing environment

INFO
- This level is more restricted than the DEBUG Java logging level
- The messages logged in this level are used to give information to the programmers
  E.g. The server has started, incoming and outgoing messages

WARN
- This level is more restricted than the INFO level
- The messages logged in this level are used to give warning messages
  E.g. Connection lost between client and server, Database connection lost etc.

ERROR
- This level is more restricted than the WARN level
- Error and exception messages are logged in this level
- ERRORS should always be printed in Java

FATAL
- This level indicates severe error conditions that leads the application to abort
- After this, the application may eventually crash or get stopped

OFF
- This level has the highest possible rank and it is intended to turn off logging in Java

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Logging using Log4j

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Log4j and Java Logging API

	log4j	java.util.logging
Manager Class	org.apache.log4j.LogManager	java.util.logging.LogManager
Logger Objects	org.apache.log4j.Logger	java.util.logging.Logger
Named Loggers	Supported	Supported
Logging Levels	• Levels are declared in the org.apache.log4j.Level class • log4j has five pre-defined levels: – FATAL – ERROR – WARN – INFO – DEBUG	• Levels are declared in the java.util.logging.Level class • There are seven pre-defined levels: – SEVERE – WARNING – INFO – CONFIG – FINE – FINER – FINEST
Log Event Object	org.apache.log4j.spi.LoggingEvent	java.util.logging.LogRecord
Output Destinations	Appender classes - these classes implement the org.apache.log4j.Appender interface	Handler classes - these classes extend java.util.logging.Handler

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Log4j and Java Logging API (Cont'd)

	log4j	java.util.logging
Output Message Filtering	Filter classes. These classes extend org.apache.log4j.spi.Filter	Filter classes. These classes must implement the java.util.logging.Filter interface
Output Message Formatting	Layout classes extend org.apache.log4j.Layout	Formatter classes extend java.util.logging.Formatter
Programmatic Configuration	Supported	Supported
Property File Configuration	Supported	Supported
XML Configuration	Supported. The XML configuration file must comply with log4j.dtd	Not supported
Email Notification	SMTPAppender class	SMTPHandler class (third-party extension)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Java Logging using Log4j

- **Given below is a sample code used to log files using Log4j**

*Invoke.java

```
2
3  import org.apache.log4j.Logger;
4
5      public class Invoke {
6          private static final Logger logger =
7             Logger.getLogger(SimpleClass.class.getName());
8          public void writeLog() {
9              logger.debug("Here Debug message");
10             logger.info("Enable logging...");
11            logger.warn("Warning Message");
12            logger.error("Here Error message");
13            logger.fatal("fatal error message");
14
15
16            try {
17               throw new Exception("Simulating an exception");
18
19
20            } catch (Exception e){
21               logger.log(null, e.getMessage());
22            }
23         }
24          public static void main(String[] args) {
25
26            SimpleClass simpleObj = new SimpleClass();
27            simpleObj.writeLog();
28            logger.info("Done...");
29         }
30
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secure Coding in Logging

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Vulnerabilities in Logging

1. It is **not** a **good practice** to log and then throw the errors

```
DemoForExceptionjava
1  import java.io.BufferedReader;
9
10 public class DemoForException {
11     private static final Logger logger =
12             Logger.getLogger(SimpleClass.class.getName(
13 public  void test()
14  {
15 try {
16     int i=5;
17     i=i/0;
18 } catch (Exception ex) {
19
20         logger.error("Error",ex);
21         throw ex;
22
23     }
24 }
25
```

2. Either log exceptions or throw exceptions; both **should not be done** together

Example: Log Exception

```
DemoForException.java
26
27 public  void Info() throws MyException
28 {
29 try {
30     int i=5;
31     i=i/0;
32 } catch (ArithmeticException ex) {
33
34         logger.error("Error",ex);
35         throw new MyException("Error",ex);
36
37     }
```

Example: Throw Exception

```
38     try {
39
40     }
41 catch (ArithmeticException ex) {
42
43     ex.printStackTrace();
44     throw new MyException("Error",ex);
45
46     }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Vulnerabilities in Logging (Cont'd)

3. Exceptions should be **thrown** instead of returning null values are returned only in a normal use case

```
DemoForException.java
47     try {
48
49     }
50 catch (ArithmeticException ex) {
51
52     ex.printStackTrace();
53     return;
54
55     }
56
```

4. Null values are returned only in a **normal use case**

```
22     public String map()
23     {try{
24
25     }catch(Exception e)
26     {
27       return null;
28     }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Logging: Vulnerable Code and Secure Code

1. Group log messages regardless of the level

- The vulnerable code below may look fine in a test case, but when it is displayed in the log file of an **application server** with **500 threads running parallel**, the two messages may be spaced up with **1000 lines** in-between in the log file

Vulnerable Code

DemoForException.java

```
57 try {
58         logger.debug("Using cache policy A");
59         logger.debug("Using retry policy B");
60     }
61 catch (ArithmeticException ex) {
62
```

Secure Code

DemoForException.java

```
57 try {
58         logger.debug("Using cache policy A, using retry policy B");
59
60     }
61 catch (ArithmeticException ex) {
62
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Logging: Vulnerable Code and Secure Code (Cont'd)

2. Parameterized logging using SLF4J

Vulnerable Code

DemoForException.java

```
57     }
58
59 try {
60         if(logger.isDebugEnabled())
61         {
62             logger.debug("created user[" + user + "] with role [" + role + "]");
63
64         }
65
66     }
67 catch (ArithmeticException ex) {
68
69     logger.error("message",ex);
70     return;
```

Secure Code

```
61
62 try {
63     slf4jLogger.debug("created user[{}] with role [{}]",user,role);
64
65     }
66 catch (ArithmeticException ex) {
67
68     slf4jLogger.error("message",ex);
69     return;
70     }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

Logging: Vulnerable Code and Secure Code (Cont'd)

3. Log the exception stacktrace

Vulnerable Code

DemoForException.java

```
57      }
58
59  try {
60      int i=5;
61      i=i/0;
62      }
63  catch (ArithmeticException ex) {
64
65      logger.error("Error in application"+ex);
66      }
67
68   }
69  }
```

Secure Code

DemoForException.java

```
57      }
58
59  try {
60      int i=5;
61      i=i/0;
62      }
63  catch (ArithmeticException ex) {
64
65      logger.error("Error in Application", ex);
66      }
67
68   }
69  }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Logging: Vulnerable Code and Secure Code (Cont'd)

4. Log exceptions in proper order

Vulnerable Code

DemoForException.java

```
33  {
34  try {
35      int i=5;
36      i=i/0;
37  } catch (Exception lolevelException) {
38
39
40      throw new ArithmeticException();
41
42      }
```

Secure Code

DemoForException.java

```
36      int i=5;
37      i=i/0;
38  } catch (Exception lolevel) {
39
40          throw new HighLevelException("Error", lolevel);
41          }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Logging: Vulnerable Code and Secure Code (Cont'd)

5. Log exceptions when they are handled. Do not log messages before they are handled

Vulnerable Code

DemoForException.java

```
35  try {
36      int i=5;
37      i=i/0;
38  } catch (Exception hklvel) {
39
40          slf4jLogger.error("catch Error herer", hklvel);
41          }
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Logging: Vulnerable Code and Secure Code (Cont'd)

6. Log uncaught exceptions

Vulnerable Code

- In web applications, uncaught exceptions can be sent to a **JSP page** and log it as follows:

web.xml

```
3 <error-page>
4 <exception-type>EmployeeNotFoundException</exception-type>
5 <location>Sorry.jsp</location>
```

Design | Source

Secure Code

- The example code is as follows:

*DemoForExcepti... | MyDomainEg.java | FileValidation.j... | DivByZeroEg.java

```
12      public static void main(String args[]) throws IOException
13      {
14
15          Thread.setDefaultUncaughtExceptionHandler(new UncaughtExceptionHandl
16              @Override
17              public void uncaughtException(Thread t, Throwable e) {
18                  Logger log = org.slf4j.LoggerFactory.getLogger(t.getClass());
19                  log.error("Exception in thread [" + t.getName() + "]", t);
20              }
21          });
22
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Secured Practices in Logging

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secured Practices in Logging

- When logging, as there is an increase in file **IO operations**, it affects the performance of the application very badly
- The following best practices should be implemented for effective Java logging:
  - Log DEBUG messages inside `isDebugEnabled()` block as given below:

Invoke.java

```
17
18          if(logger.isDebugEnabled())
19          {
20              logger.debug("Debug Log Enabled");
21          }
```

  - This practice saves a lot of string concatenation activity if the code is running in the production environment with **production logging level** instead of **debug logging level**
- Make use of good Java logging frameworks like **java.util.logging** or **log4j**
  - For example, by using **Log4j** the logging levels can be changed without restarting the application and that gives a great advantage in a production or controlled environment

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Secured Practices in Logging (Cont'd)

- Do not log sensitive data when logging in a **production environment**. Logging sensitive information may cause security risks. Example: e-mail address, phone numbers and account numbers should not be logged
- In server applications, all **bugs** and **errors** can be **tracked** only by using Java logs and too many logs will affect the application; therefore, select the appropriate logging level for every logging message
- Ensure that the format of the Java logging in the specified Java **logger** is included. In the case of **java.util.logging** API, the properties file should include the **logging formatter**. While printing logs the thread name and the Java class name should be specified
- Log messages consistently and the messages must be **informative**. The data should be **printed** with the **message** wherever required
- Use code level prefix when logging information (e.g. **database code** or **session code** etc.) so that it will be clear to identify which part of the code is **printing** the log message

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Module Summary

- Exception and error occur when an unexpected or abnormal event takes place during the execution of the application
- Exceptions in Java are handled using 'try-catch-finally' blocks
- java.lang.InterruptedException is thrown when a thread is interrupted while sleeping or waiting
- Use of return, break, continue or throw statements is restricted in a finally block
- System.exit() when invoked can even terminate the Java Virtual Machine (JVM) resulting in the termination currently running programs as well as threads that might even lead to denial-of-service (DoS) attacks
- RuntimeException, Exception, or Throwable should not be thrown or else it may lead to various errors
- printStackTrace() function should not be used in code as it reveals all the exception information to the user
- In Java, logging frameworks are provided from package java.util.logging and used to write log messages on to a central location

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

JAVA

C | ASE

Certified | Application Security Engineer



Module 09

Static and Dynamic Application Security Testing (SAST & DAST)

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

Module Objectives

1. Introduction to Static Application Security Testing (SAST)
2. Discuss Manual Secure Code Review Techniques for Most Common Vulnerabilities
3. Introduction to Dynamic Application Security Testing
4. Discuss DAST using Automated Application Vulnerability Scanning Tools
5. Discuss DAST using Proxy-based Security Testing Tools

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Static Application Security Testing

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Static Application Security Testing (SAST)

1. SAST, also known as **secure code review**, is one of the software security assurance approach to identify security-related **weaknesses** in the code
2. It involves detailed systematic inspection of **source code** to detect vulnerabilities and design flaws
3. It should be performed toward the end of the source code development when application code is **stable** or **nearly completed**
4. It should always be performed in combination with **human effort (Manual)** and **technology support (Automated)**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Objectives of SAST

- SAST aims to detect **application security vulnerabilities** and their **root causes** when code is not running

Root cause analysis (Source to sink)

Source to Sink analysis (what is the source (input) and the sink (vulnerable code pattern) for any vulnerability) is carried out using **Data flow diagrams**

Design Analysis

Design flaws like **mishandling of nonuser inputs** and external integrations like **server-to-server** can be easily identified

Find all Instances of the Common Vulnerabilities

Security code review enumerates all instances of **certain vulnerability**

Identify Uncommon Security Flaws

Security code review being specific to the application may **highlight some flaws** like insecure termination of execution flow, synchronization error, etc.

Analyze Effectiveness of Existing Security Controls

Security code review enumerates **weakness** in the existing security control against known **threats**

Effective Remediation Recommendations

Suggests **mitigation techniques** that can best suit the application instead of a generic one

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Why SAST

- Many serious **security vulnerabilities** can be **addressed**
- Decreases the remediation **cost** of **vulnerabilities** in later stages
- Deeper analysis is possible with **less false positive** and **negative**
- Help you to add best secure coding practices as per **department coding standards** which can prevent future security issues

Skills Required for SAST

1. Able to think from the attacker's perspective
2. Working knowledge of the **programming language** under review
3. Knowledge of standard, secure coding techniques
4. Quickly able to **detect insecure coding** practices
5. Analytical skills

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

What to Look for in SAST

- Look for the code that implements common application **security mechanisms**
- The **insecure** implementation of security mechanisms can render applications vulnerable to various attacks

Common Application Security Mechanisms

1. Authentication
2. Authorization
3. Session Management
4. Data Validation
5. Error Handling
6. Logging
7. Encryption
8. Security Configuration

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Common Vulnerabilities Identified through SAST

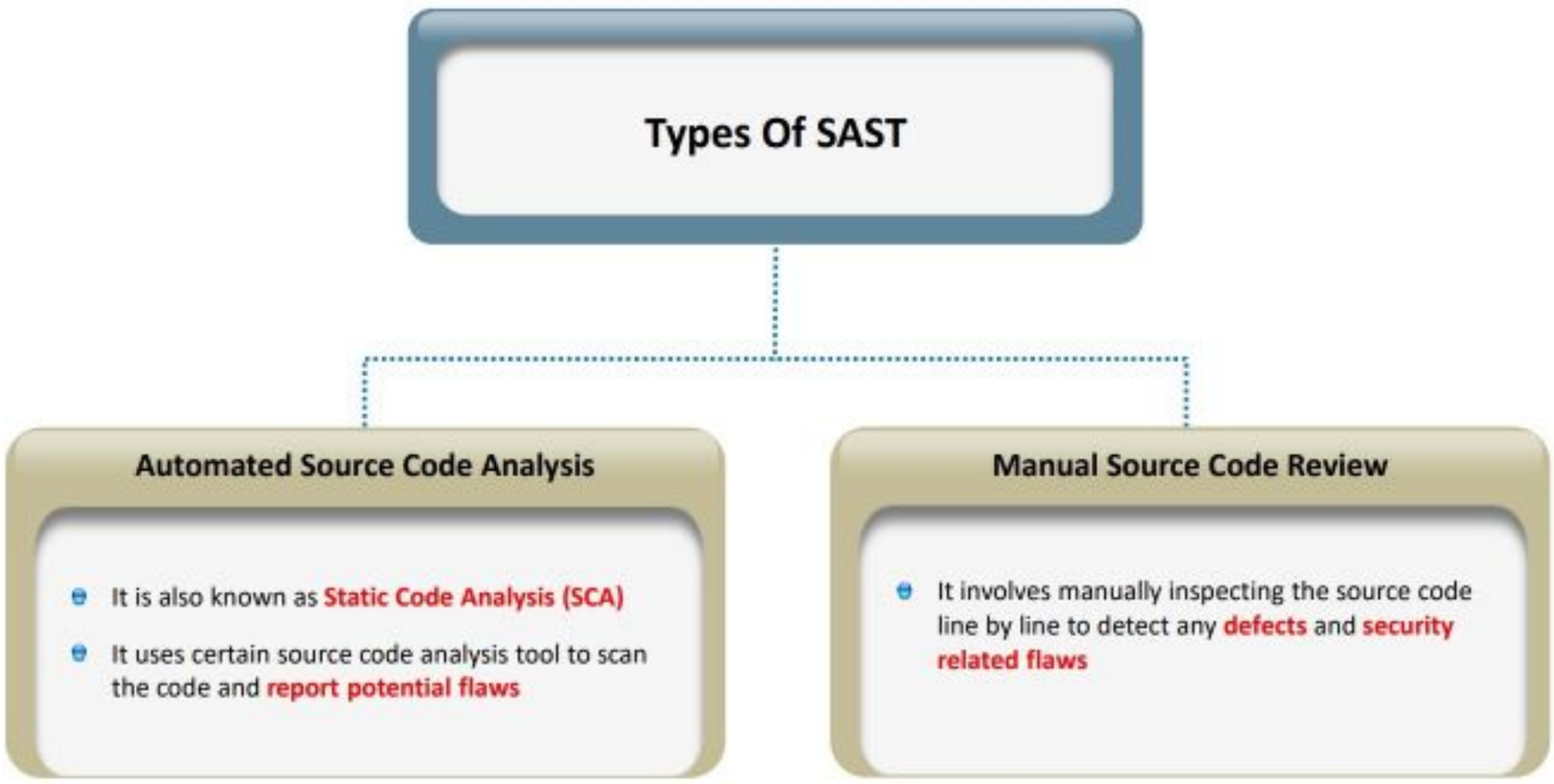
1. SQL Injection
2. Cross-site Scripting
3. Authentication Flaws
4. Cryptographic Flaws
5. Buffer Overflows
6. Malicious File Execution
7. Cross-site Request Forgery
8. Sensitive Information Leakage

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

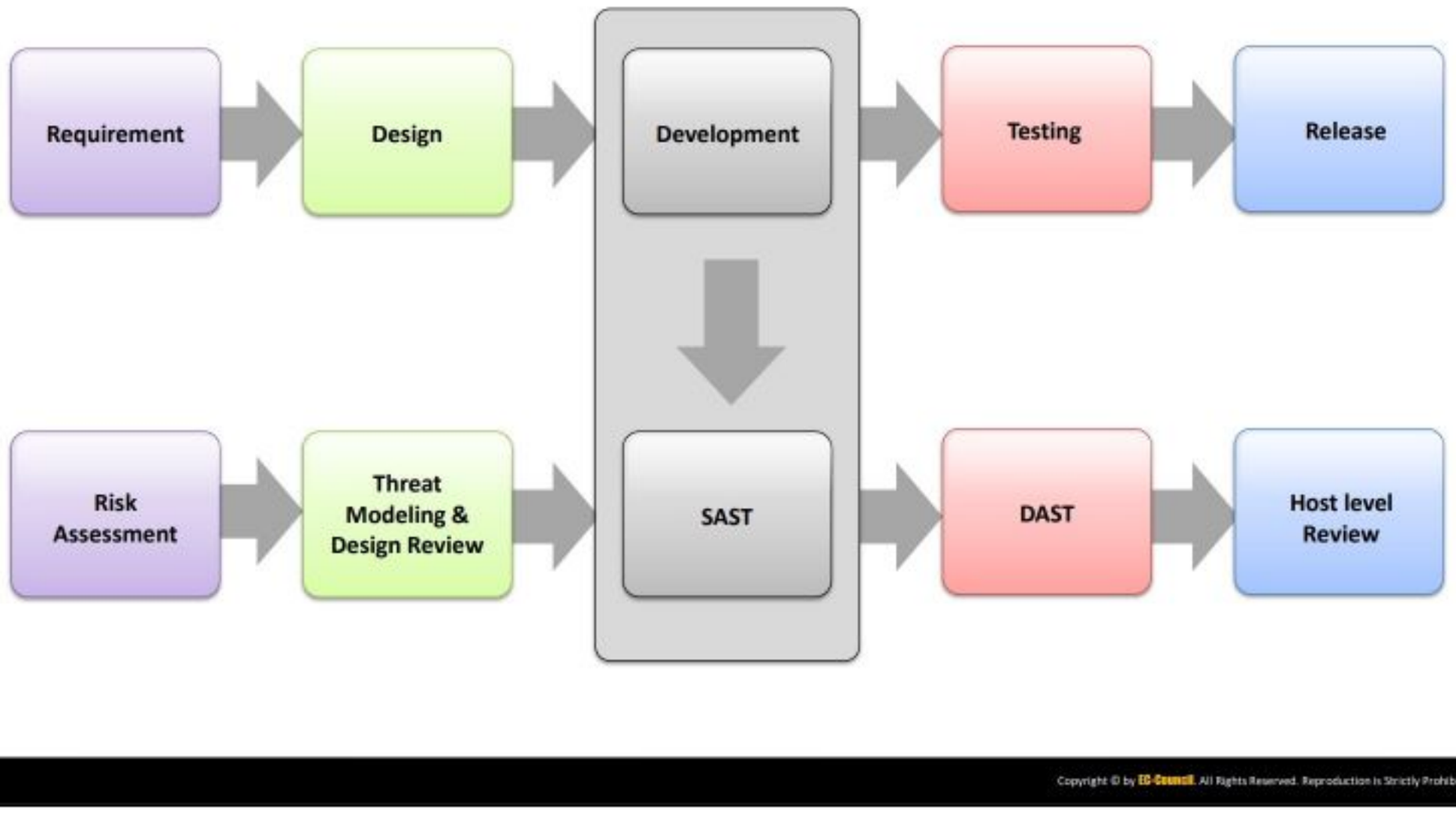
@ErrorCatchers

Types of SAST



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Where does SAST Fit in SDLC?



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

SAST Steps

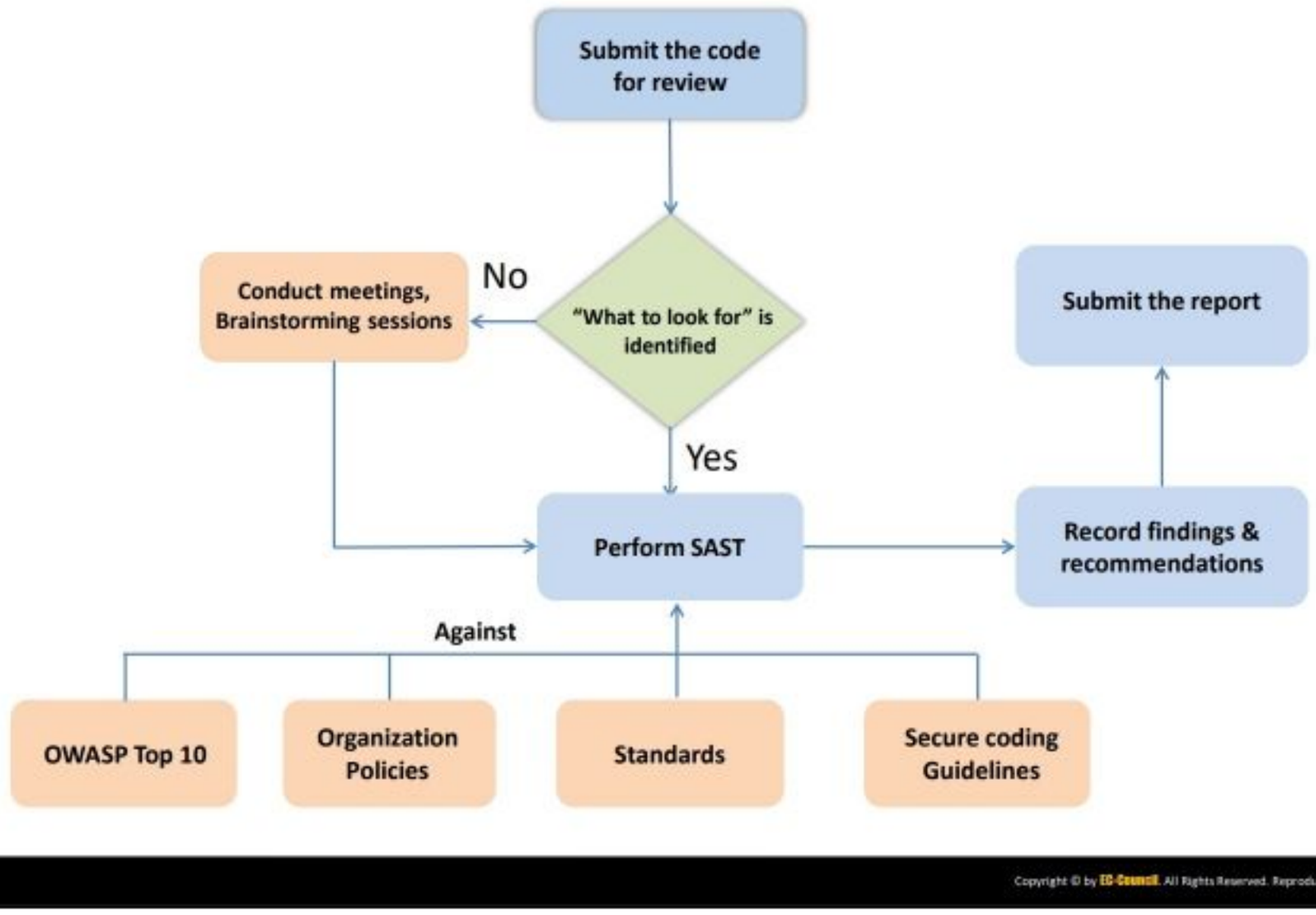
1. **Perform an application walkthrough**
   - Understand the applications functionality
2. **Developers Interview**
   - Talk to **developers**; more you involve developers, more effective you secure code review will be
   - It will help you quickly
     - Determine the **high risk areas**
     - Understand **developer's logic** behind particular functionality
     - Establish **friendly** and **respectful** relationship between you and developers
3. **Analyze threat Model**
   - Figure out the major existing **security threats**
4. **Define secure code review objectives**
   - Based on threat model analysis, identify the objectives such as:
     - Code review for all major **security vulnerabilities**
     - Code review for **OWASP top 10**
     - Code review for **specific vulnerability**



5. **Define scope for secure code review**
   - Be clear on what to **look** for and what to **avoid**
   - Calculate the **time** and **efforts** required
   - Define the **budget constraints**
6. **Categorized the Objectives**
   - Prioritize the objective based on **severity** and **requires more attention**
7. **Conduct Static Code Analysis**
   - Use **multiple** source code analysis tools to perform secure source code analysis
   - Most common vulnerabilities and flaws are **identified**
8. **Conduct Manual Source code Review**
   - Inspect the application's code line by line in order to find the **use of insecure coding practices**
9. **Remediation/Recommendation**
   - Suggest secure coding practice for each **findings**
10. **Prepare Report**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SAST Activities-flow Chart



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Recommendation for Effective SAST



01 Code review should always be done by a **third party person**, a **peer** or an **external entity**

02 Understand the application architecture and business logic and sort out the **possible security issues** that the application can be vulnerable to

03 Set clear set of objectives for "**what to look for**" based on security issues

04 Review code for security objectives only. Do not review code for other issues like performance, business logic, reliability, etc. Do it **separately**

05 Spend time to set **clear objectives**

06 Prioritize the security issues; higher-level security vulnerabilities should be **addressed first**

07 Set a **reasonable time limit** for the review

08 Do not lose track while going into detail of the **higher-level** security vulnerabilities

09 Do not spend too much time in not a high-priority area or objective, **address them later**

10 Always select small and manageable piece of code for review. It will help you to find a larger number of security vulnerabilities with **great focus** and within **less time**

11 Use multi-pass approach. Let the multiple people allow to review the **same code**

12 Detect secure coding mistakes that are repeatedly committed and add the corresponding secure coding practice in your **development department's coding standards**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SAST Deliverable



- **SAST report should include:**

1. Types of vulnerabilities
2. Severity level/Impact
3. Location, line of code where security issue exists
4. Recommendations

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Automated Source Code Analysis

- With the help of automated source code analysis, you can **identify** the code associated with most common application **security vulnerabilities**
- The **static code analysis** tools automate the entire process of secure code review; however, they scans the code **superficially**
- Multiple static code analysis tools should be used to conduct **secure code review**

Advantages:

- Takes **very less time** to complete

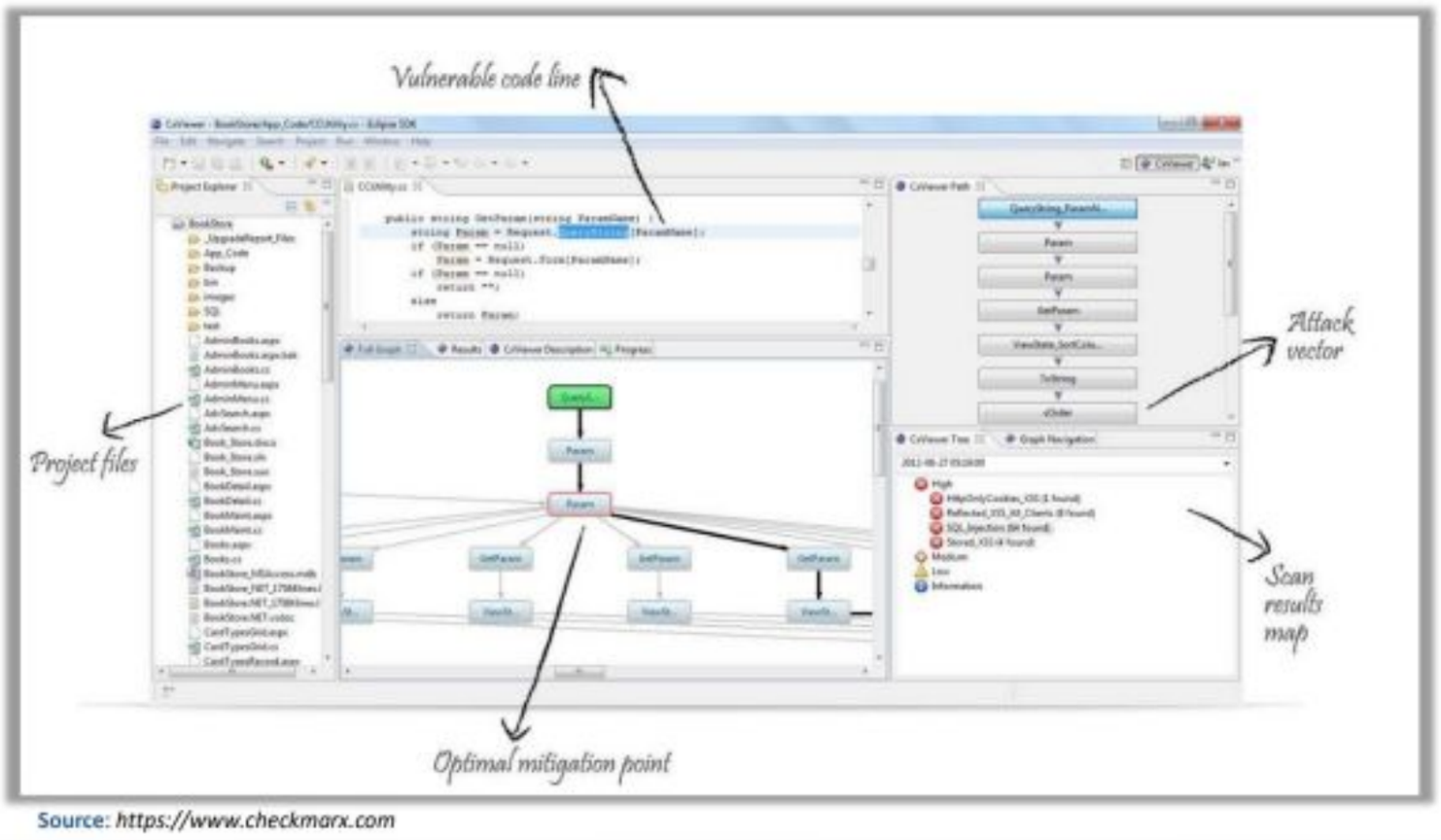
Disadvantages:

- Unable to **uncover** every issue
- Good automated review tools are **expensive**
- High possibility of **false-positive** or **false-negative errors**
- Requires **human intervention** to deal with of false-positive or false-negative

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Static Code Analysis using Checkmarx Static Code Analysis

- **Checkmarx** static code analysis software, seamlessly integrated into development process which identifies hundreds of known code vulnerabilities



Source: *https://www.checkmarx.com*

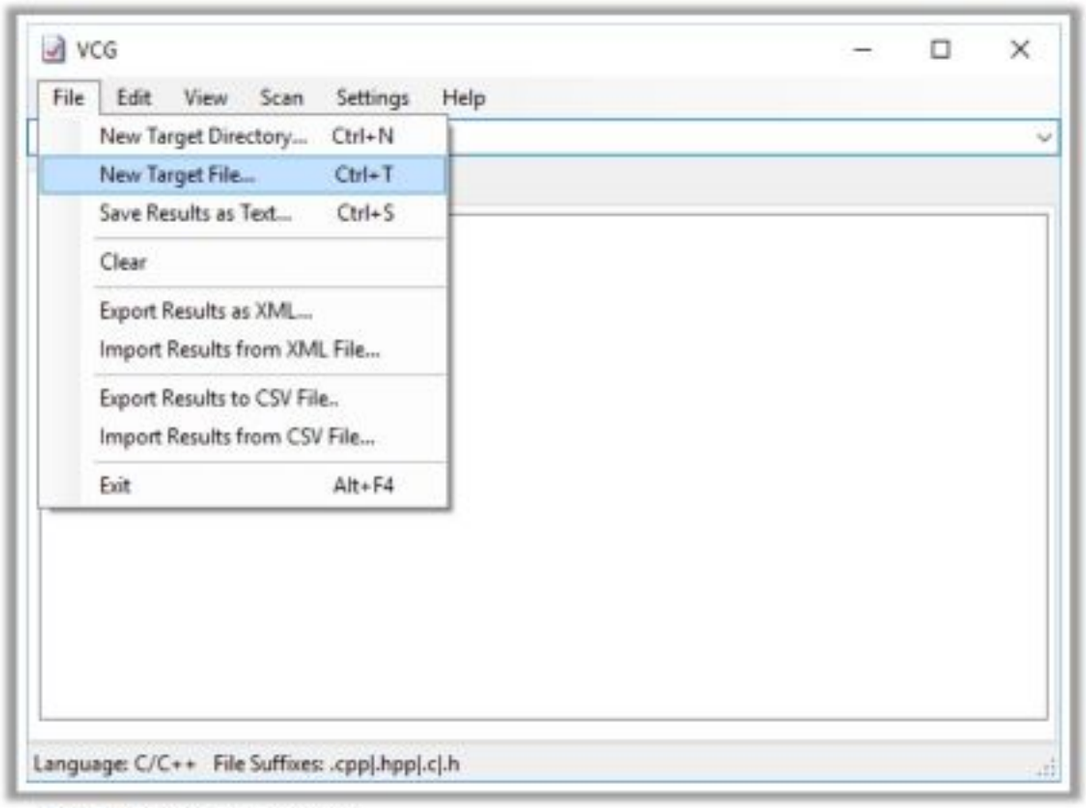
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Static Code Analysis using Visual Code Grepper (VCG)

- VCG is an automated code security review tool for C++, C#, VB, PHP, Java and PL/SQL which is intended to drastically speed up the code review process by **identifying bad/insecure code**
- Steps for SCA:
  - **Install** and **run** the VCG tool
  - Select the **language** of your project
  - Browse the **project/file** of your interest and click **Scan**

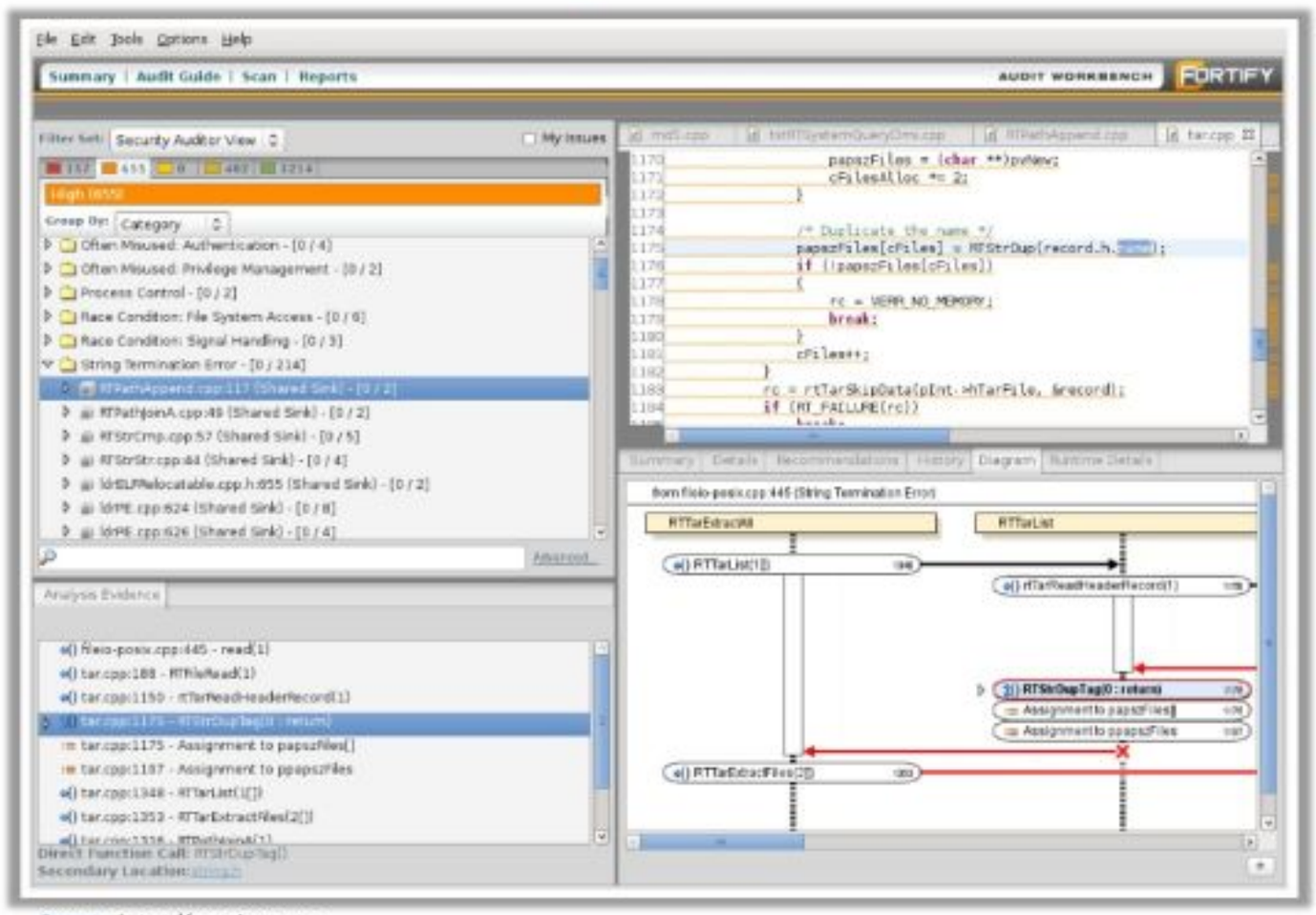


Source: *https://sourceforge.net*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Static Code Analysis using HP Fortify

- Fortify Static Code Analyzer is an **automated static code analysis** to help developers eliminate vulnerabilities and build secure software



Source: *https://saas.hpe.com*

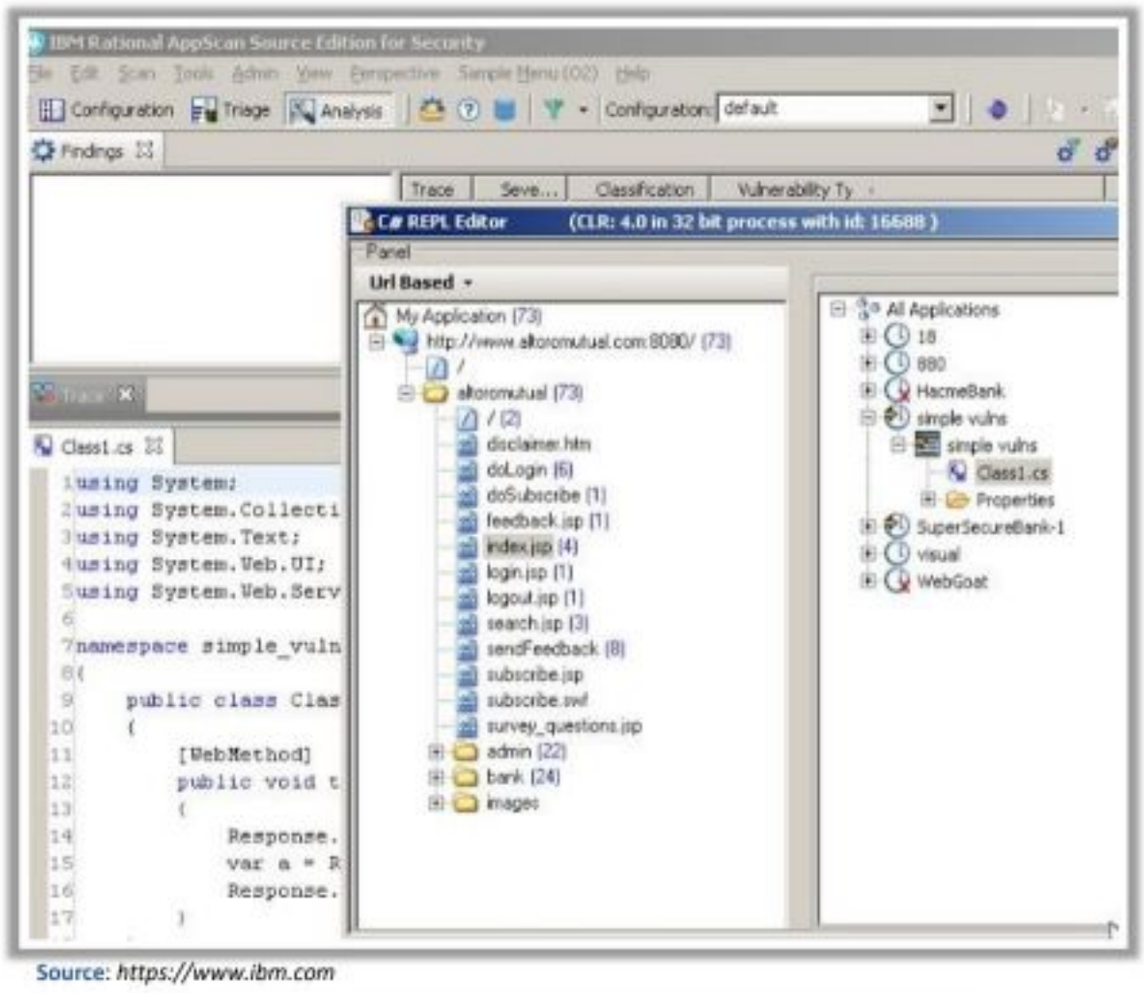
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Static Code Analysis using Rational AppScan Source Edition

- AppScan Source is a static application security testing (SAST) solution which scans your application source code for **security vulnerabilities**



Source: *https://www.ibm.com*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Selecting a Static Analysis Tool

- You need to select the right tool while selecting the static analysis tool for secure code review
- The process of selecting the **right tool** depends on certain conditions:
  - The static analysis tool used for reviewing the code should support the **programming language** used for developing the application
  - The accuracy of the tool with respect to **false-negative** and **false-positive errors**
  - The tool needs to understand the **libraries** and **frameworks** used in the application
  - It should be easy to **integrate the tool** to the IDE selected for developing the application
  - The **time** and **effort** required for the set-up of the tool
  - The type of analysis preferred for code review process
  - The license cost incurred for the tool
- You should have a clear idea regarding the type of static code analysis required for performing **code review**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Manual Secure Code Review

With manual secure code review, you can analyze and detect many security vulnerabilities that are missed out during the **static code analysis**

Advantages

- Less chance of getting **false-positive** and **false-negatives**
- High percentage of **vulnerabilities** can be detected

Disadvantages

- Time consuming process
- Requires **good understanding** of the code used

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Manual Secure Code Review for Most Common Vulnerabilities

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for PCI DSS Compliance

- The PCI DSS compliancy is checked against a list of **PCI DSS requirements**
- Check the applications against a list of PCI DSS requirements

PCI DSS Requirement	Description
6.3.7	Review of the custom code for vulnerabilities before releasing the code to production or customers
6.3.5	Deletion of custom user accounts, usernames and passwords before releasing it to the customers
6.5	Review and prevention of common coding vulnerabilities in an application
6.6	Review of custom application codes and installation of application layer firewall for web applications

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for PCI DSS Compliance (Cont'd)

- **The PCI DSS requirement 6.5 reviews and prevents common coding vulnerabilities such as:**

- 6.5.1 Unvalidated input
- 6.5.2 Broken access control
- 6.5.3 Broken authentication and session management
- 6.5.4 Cross-site scripting (XSS) attacks
- 6.5.5 Buffer overflows
- 6.5.6 Injection flaws
- 6.5.7 Improper error handling
- 6.5.8 Insecure storage
- 6.5.9 Denial of service
- 6.5.10 Insecure configuration management

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Blacklisting Validation Approach

- **Code Pattern**: Look for code where the developer is attempting to use a **blacklisting approach** to validate user inputs

```
Source Navigate Search Project Run Window Help
*Login.jsp   ValidateUser.java
1  package Validation;
2
3  public class ValidateUser {
4
5
6  int valiuser()
7  {
8      String[] userInputs= new String[]{"<SCRIPT>","SELECT","UNION","WHERE","</SCRIPT>","--",">","<"};
9      String userName =getuserName();
10     if(userName.toUpperCase().contains(userInputs.toString()))
11     {
12
```

- **Recommendation**: Use a **whitelisting** based validation approach to validate your inputs

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for Client-side Validation Approach

- **Code Pattern**: Look for code where the developer is attempting to use only **client-side input validation script** to validate user inputs

```
Edit Source Refactor Navigate Search Project Text Run Window Help
*register.jsp
119                                  </label>
120                                </div>
121                              </div>
122                            </div>
123                <div class="col-lg-12  col-md-12 col-sm-12">
124    <input type="submit" ID="Btn_Register"  class="btn  submitButton"  value="Register" onClick="return ValidateUser();" >
125                </div>
126              </form>
127            </div>
128            <br/>
129            <br/>
130          </div>
131          <div class="col-lg-3 col-m
132    </div>
```

```
Edit Source Refactor Navigate Search Project Text Run Window Help
*register.jsp
11     <script>
12       function ValidateUser() {
13           var Name,pwd, gender, conpwd, EmailId, emailExp;
14           Name = document.getElementById("txtUserName").value;
15           gender = document.getElementById("ddlType").value;
16           pwd = document.getElementById("txtpwd").value;
17           conpwd = document.getElementById("txtcpwd").value;
18           EmailId = document.getElementById("txtmail").value;
19           emailExp = /^([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([com\co\.\in])+$/; // to validate email id
20           if (Name == '' && gender == 0 && pwd== '' && conpwd == '' && EmailId == '') {
21               alert("Enter All Fields");
22               return false;
23           }
```

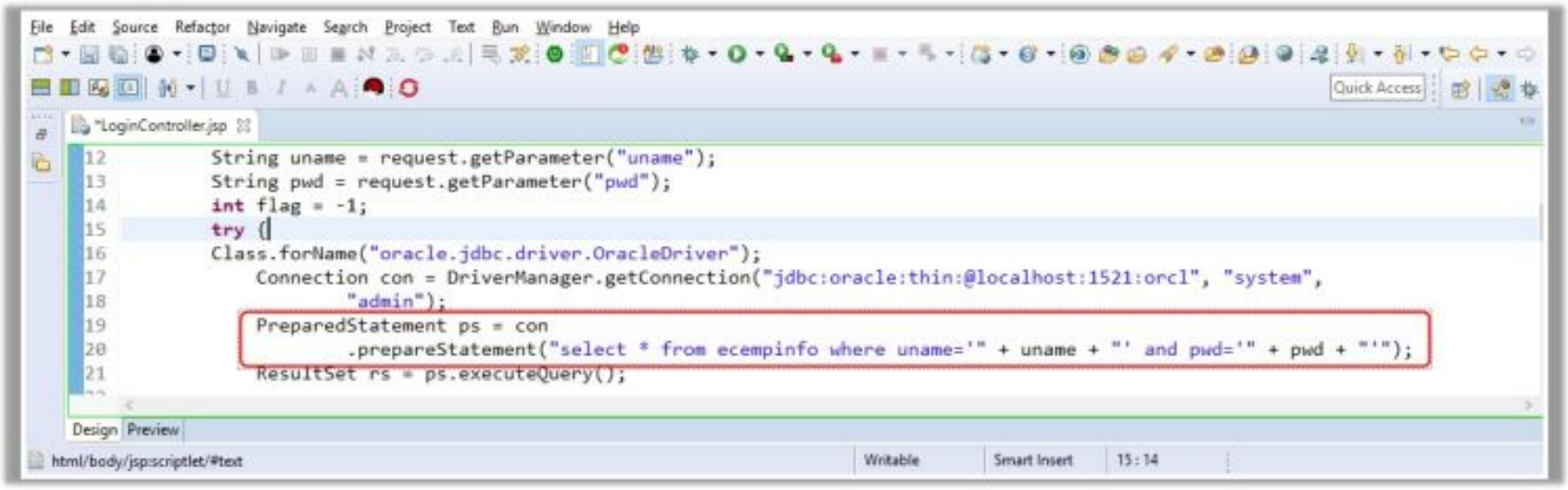
- **Recommendation**: Client-side validation can be bypassed easily. Use both **client** and **server-side validation** techniques to validate user inputs

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Non-parametrized SQL Query

- **Code Pattern**: Look for code where the developer is attempting to write a **non-parametrized query**

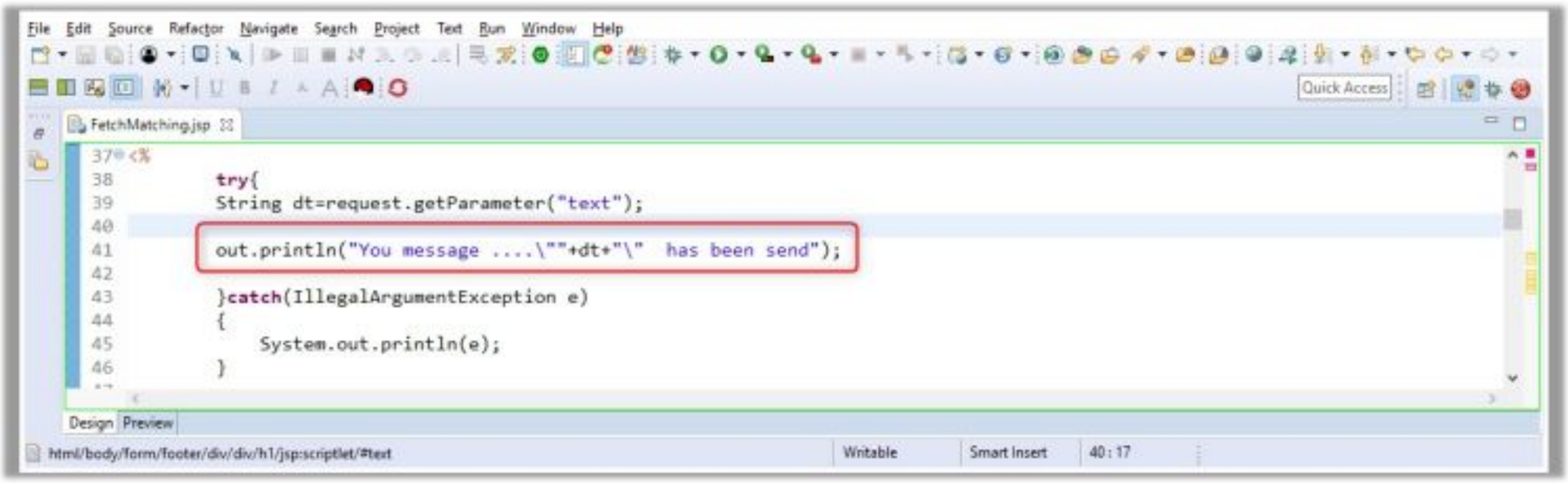


Recommendation: Always use **parametrized** query to avoid **SQL injection attacks**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for XSS Vulnerability

- **Code Pattern**: Look for code where the developer is attempting to **render output without encoding**



Recommendation: Always **encode** your output to avoid **XSS**

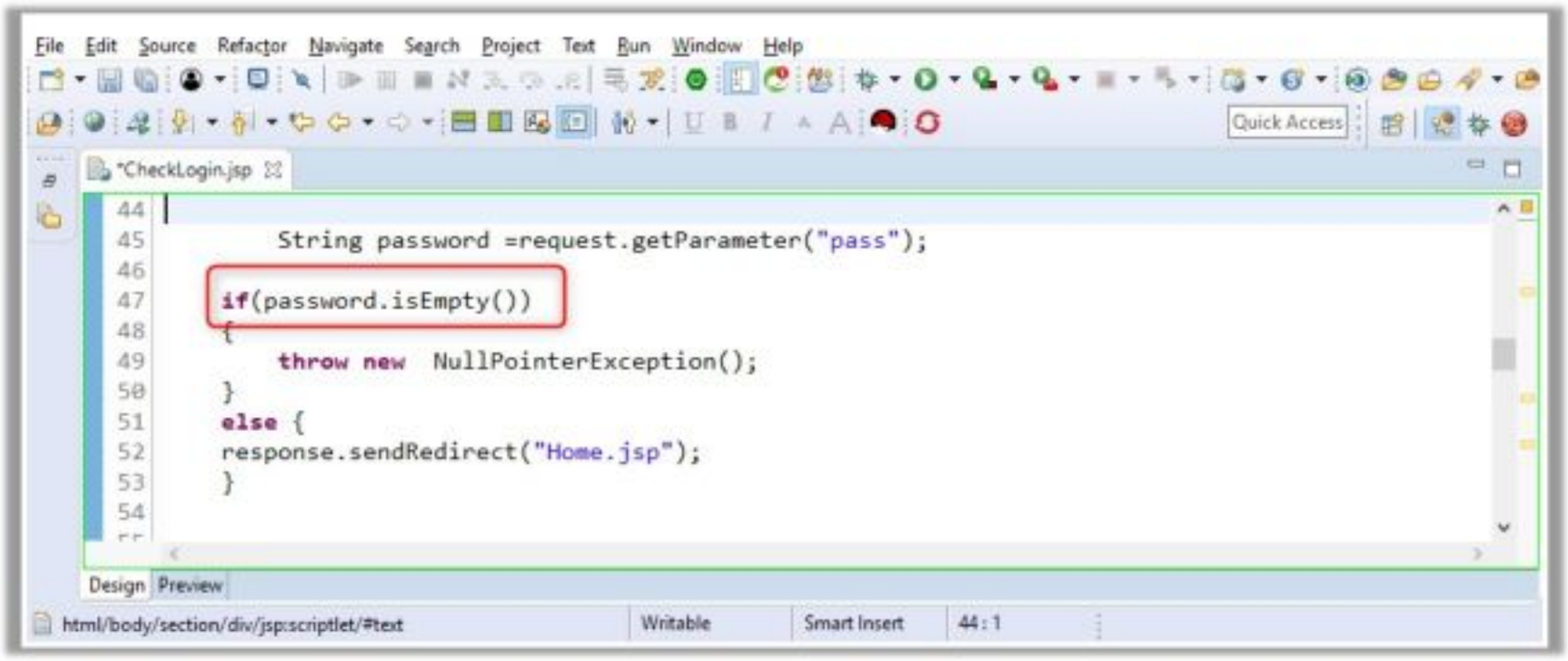
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Weak Password Authentication

- **Code Pattern**: Look for code where the developer is attempting to authenticate users with a **weak password condition**

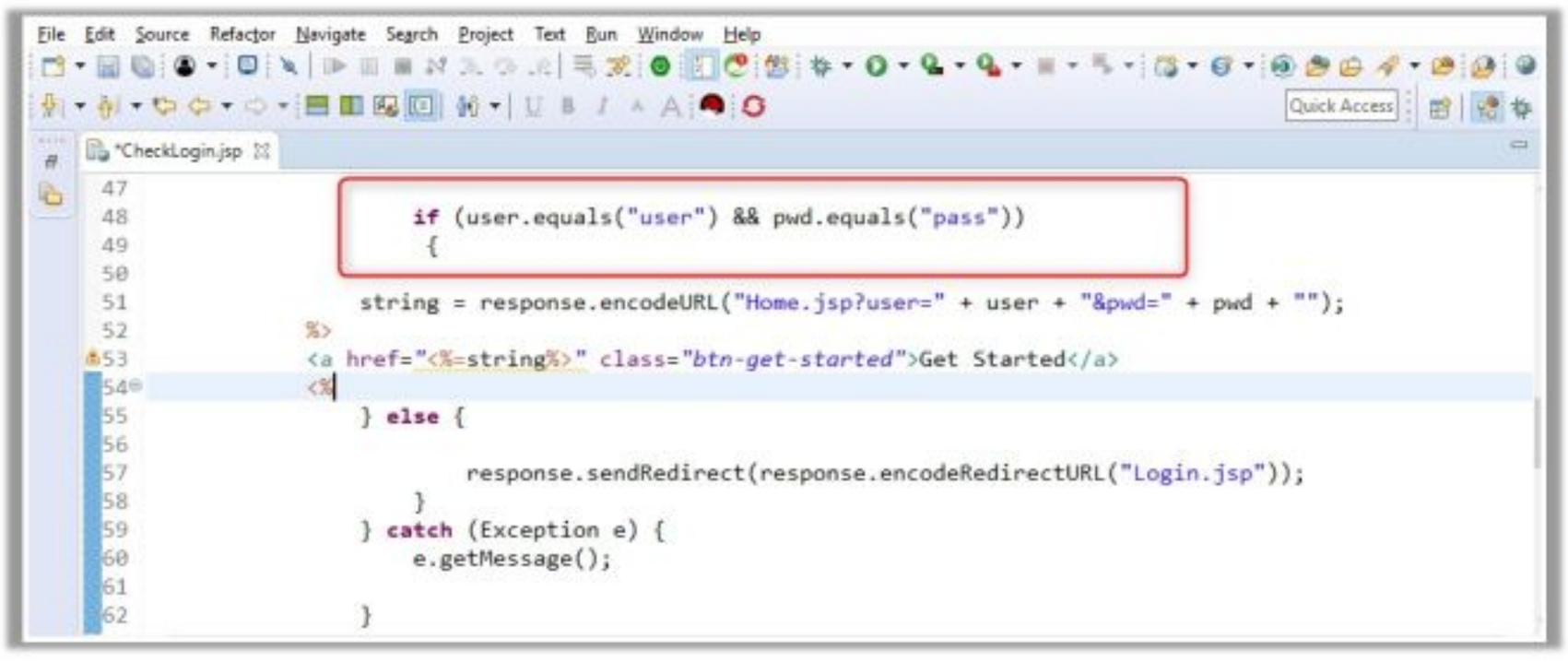


Recommendation: Always use **strong password rules** or **conditions** to authenticate users

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for Hard-coded Passwords

- **Code pattern**: Look for code where the developer is attempting to use **hard-coded passwords**



Recommendation: Never use hard-coded secrets such as **passwords** in coding. Hard-coded passwords can be an easy target for attackers

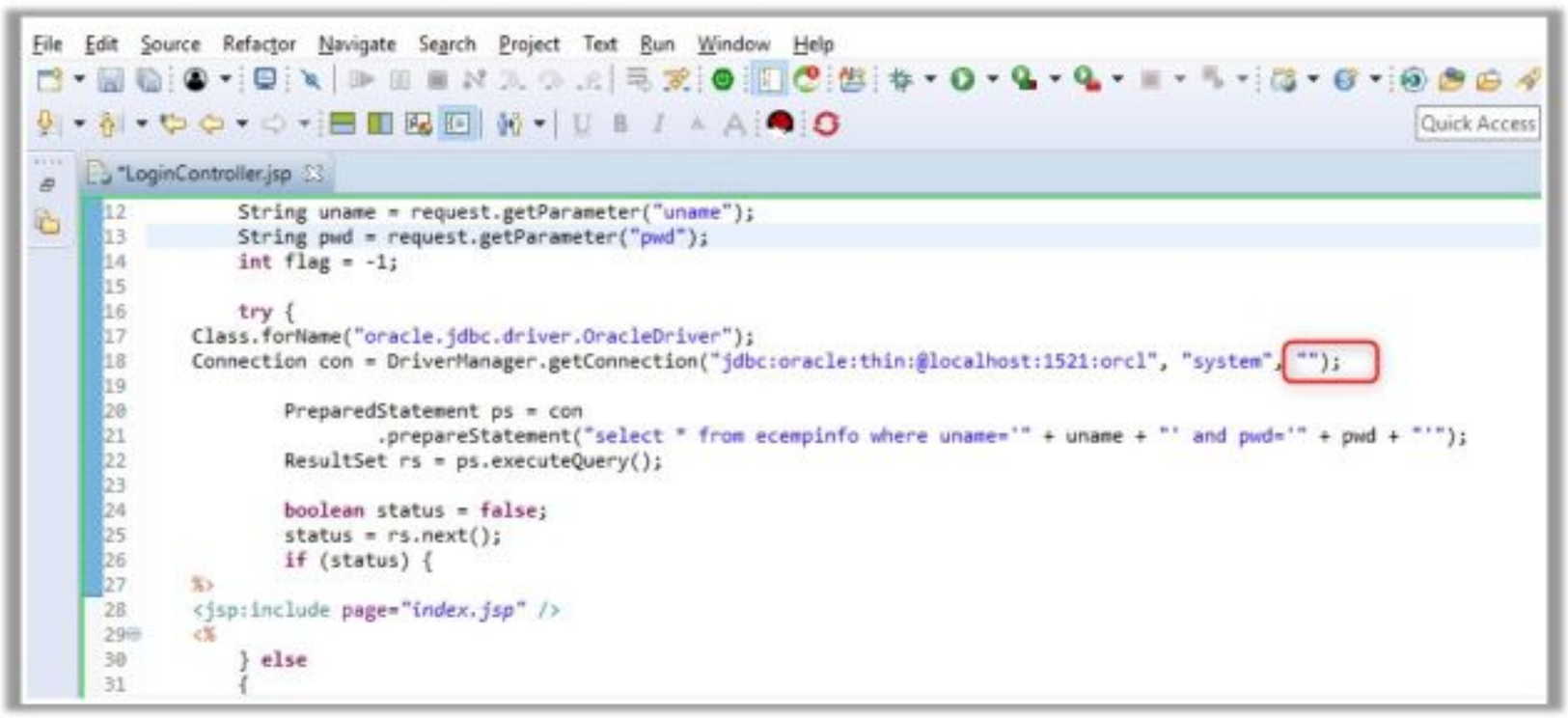
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Empty Password in Connection String

- **Code pattern**: Look for the connection string where the developer is attempting to use an **empty password** to connect to a remote database server



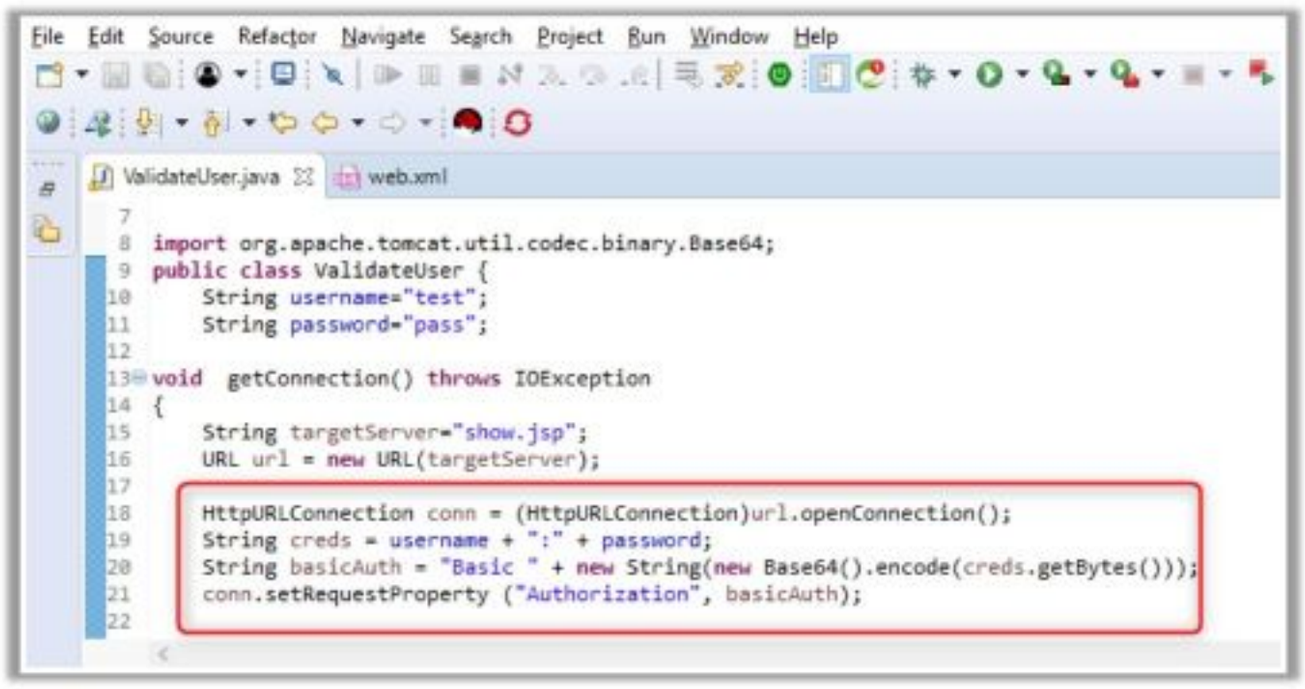
```
12          String uname = request.getParameter("uname");
13          String pwd = request.getParameter("pwd");
14          int flag = -1;
15
16          try {
17      Class.forName("oracle.jdbc.driver.OracleDriver");
18      Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "system", "");
19
20              PreparedStatement ps = con
21                      .prepareStatement("select * from ecempinfo where uname='" + uname + "' and pwd='" + pwd + "'");
22              ResultSet rs = ps.executeQuery();
23
24              boolean status = false;
25              status = rs.next();
26              if (status) {
27      %>
28      <jsp:include page="index.jsp" />
29      <%
30          } else
31          {
```

Recommendation: Never set **empty** or **plain text** for connection string as this will facilitate **dictionary attacks** or **brute force attacks**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for Insecure Basic Authentication

- **Code Pattern**: Look for code where the developer is attempting to use **basic authentication and HTTP connection** without using SSL



```
7
8   import org.apache.tomcat.util.codec.binary.Base64;
9   public class ValidateUser {
10      String username="test";
11      String password="pass";
12
13  void  getConnection() throws IOException
14  {
15      String targetServer="show.jsp";
16      URL url = new URL(targetServer);
17
18      HttpURLConnection conn = (HttpURLConnection)url.openConnection();
19      String creds = username + ":" + password;
20      String basicAuth = "Basic " + new String(new Base64().encode(creds.getBytes()));
21      conn.setRequestProperty ("Authorization", basicAuth);
22
```

Recommendation: Use **SSL** for basic authentication so that credential can not be decoded easily

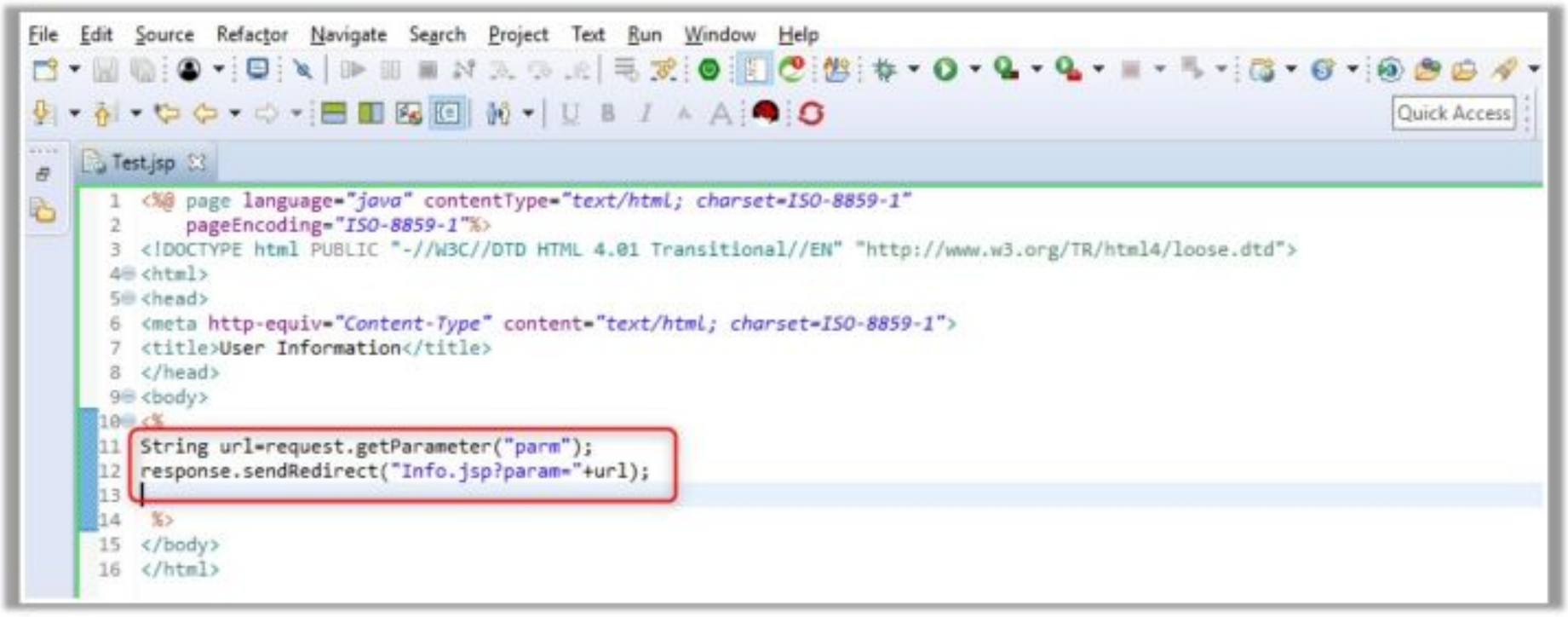
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Open Redirect

- **Code Pattern**: Look for code where the developer is attempting to **redirect** to the path that is created from user supplied input

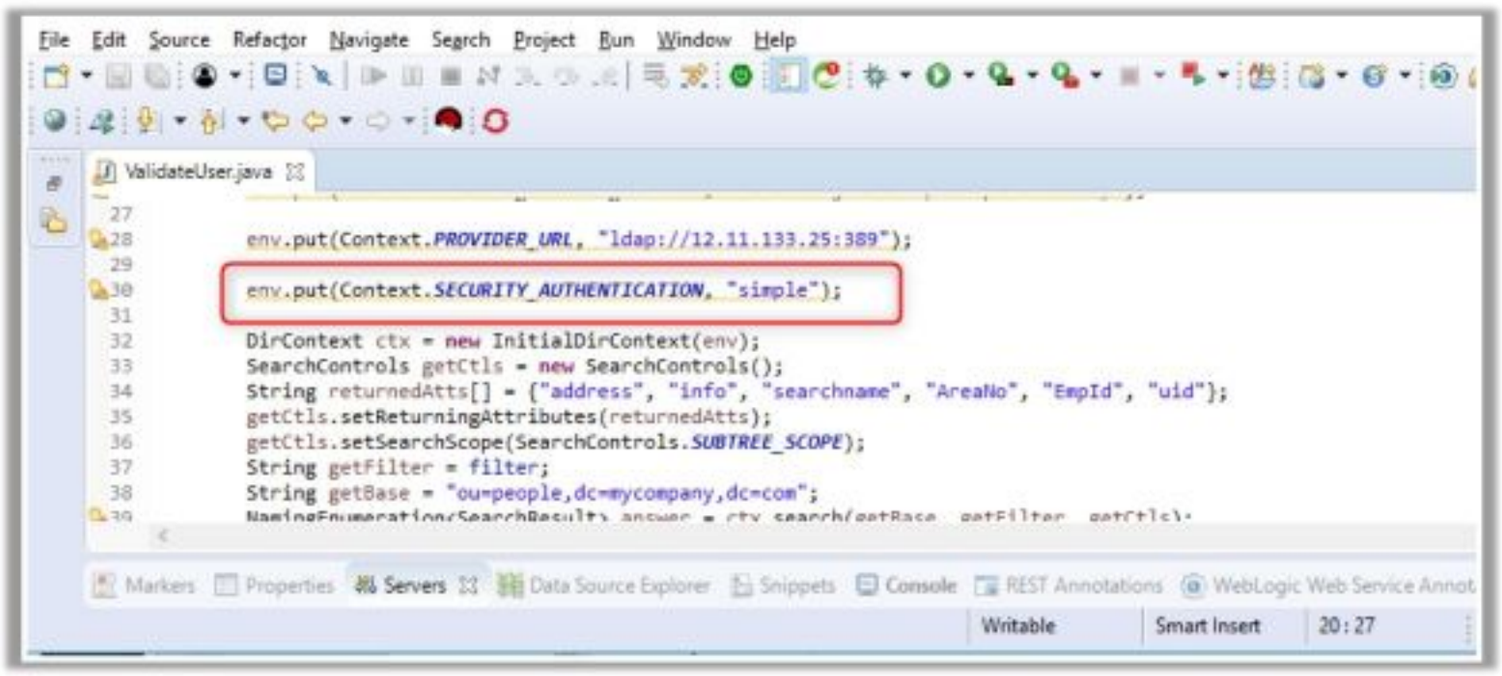


- **Recommendation**: Always avoid using **redirects** or **forwards.** User input should not be used to construct the source path. If it is necessary to send a parameter then check if the parameter is valid. Ensure that the user is authorized to browse the constructed URL

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for Insecure LDAP Authentication

- **Code Pattern**: Look for code where the developer is attempting to use **"simple" basic authentication** which is weak; attackers can easily get cleartext



- **Recommendation**: Always use secure mechanisms like those listed in LDAP. LDAP supports **SASL** mechanism, SASL provides a security layer for authentication between client and server

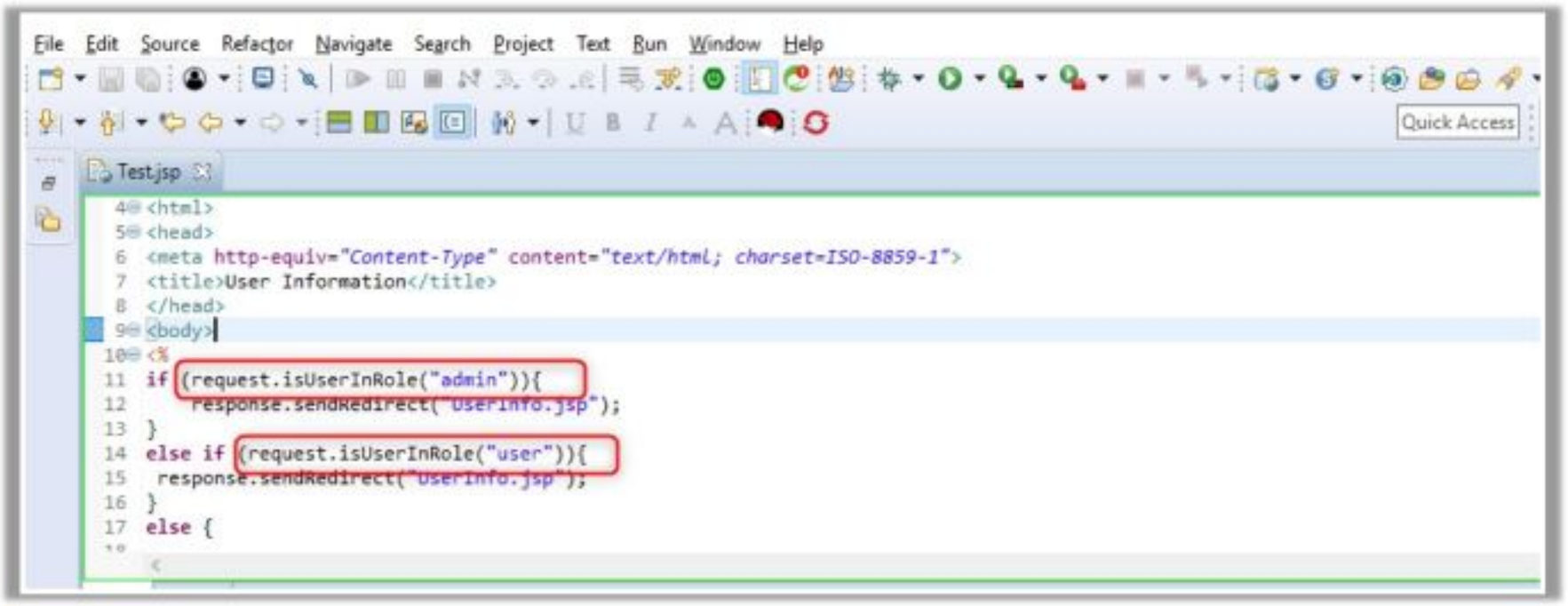
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Insecure Authorization Mechanism

- **Code Pattern**: Look for code where the developer is attempting to use an **inadequate authorization mechanism** by which the code becomes complex and also hard to maintain. Even if the mistake is simple, an attacker can easily abuse the code

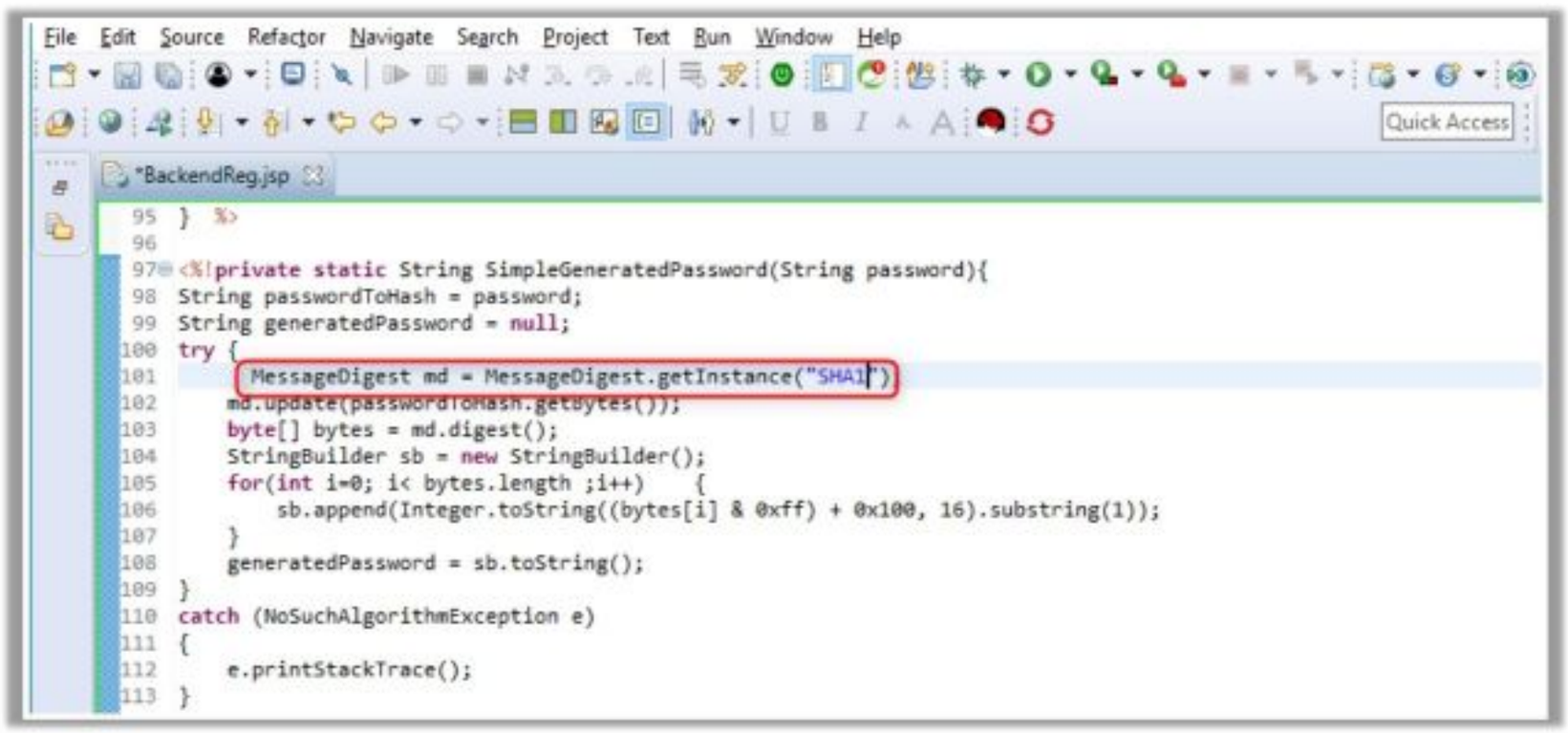


- **Recommendation**: Always use a **proper authorization** method which is secure and easy to be altered

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for Weak Password Length

- **Code pattern:** Look for code where the developer is attempting to set passwords with a **weak hashing** algorithm



- **Recommendation**: Set passwords with a strong hashing function like **Bcrypt** to make it unbreakable

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Use of Weak Hashing Algorithm

- **Code Pattern**: Look for code where the developer is attempting to set a **weak hashing algorithm**

```
95  } %>
96
97  <%!private static String SimpleGeneratedPassword(String password){
98  String passwordToHash = password;
99  String generatedPassword = null;
100 try {
101      MessageDigest md = MessageDigest.getInstance("MD5");
102      md.update(passwordToHash.getBytes());
103      byte[] bytes = md.digest();
104      StringBuilder sb = new StringBuilder();
105      for(int i=0; i< bytes.length ;i++)    {
106          sb.append(Integer.toString((bytes[i] & 0xff) + 0x100, 16).substring(1));
107      }
108      generatedPassword = sb.toString();
109  }
110 catch (NoSuchAlgorithmException e)
111 {
112      e.printStackTrace();
113 }
```

- **Recommendation**: Always use **strongest hashing algorithm**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for Use of Weak Random Number Generator

- **Code Pattern**: Look for code where the developer is using a **weak random number generation method.** An 8 character string can be predicted by an attacker

```
24      String symbols = "ABCDEFGHIJKLMNOPQRYZabcdefghijklmxyz0123456789";
25
26      Random random = new Random();
27      char[] buffer = new char[8];
28      for (int i = 0; i < buffer.length; ++i)
29      {
30      buffer[i] = symbols.charAt(random.nextInt(symbols.length()));
31      }
32      String randomString = new String(buffer);
33
34
```

- **Recommendation**: To prevent an attacker from predicting random numbers, use a string with more than **8 characters** to generate random numbers

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Use of Insecure PBE Work Factor

- **Code Pattern**: Look for code the developer is using with a **work factor** of **5000** which is considered to be insecure

```
25 public void uinfo()
26 {
27
28     String password="";
29     String salt="";
30     try {
31     char[] passwordChars = password.toCharArray();
32     byte[] saltBytes = salt.getBytes();
33
34     PBEKeySpec spec = new PBEKeySpec(passwordChars, saltBytes, 5000, KEY_LENGTH);
35
36     SecretKeyFactory key = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
37     byte[] hashedPassword = key.generateSecret(spec).getEncoded();
```

Recommendation: The developer should use **big iteration value** which generates complex hashing to prevent the password from being predicted

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for Use of Weak Encryption Algorithm

- **Code Pattern**: Look for code where the developer is attempting to set a **weak encryption algorithm**

```
1  package Validation;
2  import javax.crypto.Cipher;
3  import javax.crypto.spec.SecretKeySpec;
4  public class ValidateUser {
5      public static String userDataEncrypt(String userClearText,String userKey) throws Exception{
6          String userInfo="";
7
8          try {
9              SecretKeySpec skeyspec=new SecretKeySpec(userKey.getBytes(),"DES");
10             Cipher cipher=Cipher.getInstance("DES");
11             cipher.init(Cipher.ENCRYPT_MODE, skeyspec);
12             byte[] encrypted=cipher.doFinal(userClearText.getBytes());
13             userInfo=new String(encrypted);
14
15         } catch (Exception e) {
16             e.printStackTrace();
17             throw new Exception(e);
18         }
19         return userInfo;
20     }
21 }
```

Recommendation: Always use the **strongest encryption algorithm**

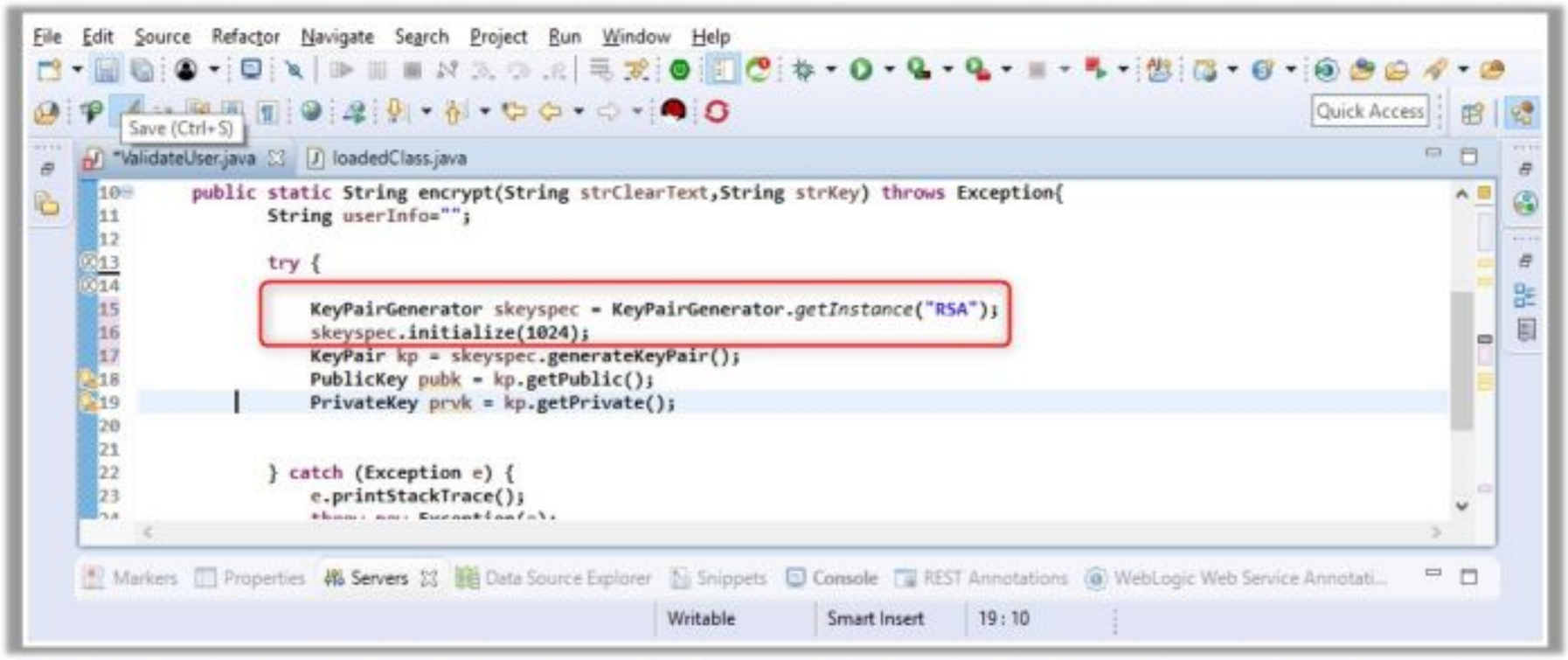
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Use of Insufficient Encryption Key Size

- **Code Pattern**: Look for code where the developer is attempting to set an insufficient encryption **key size**

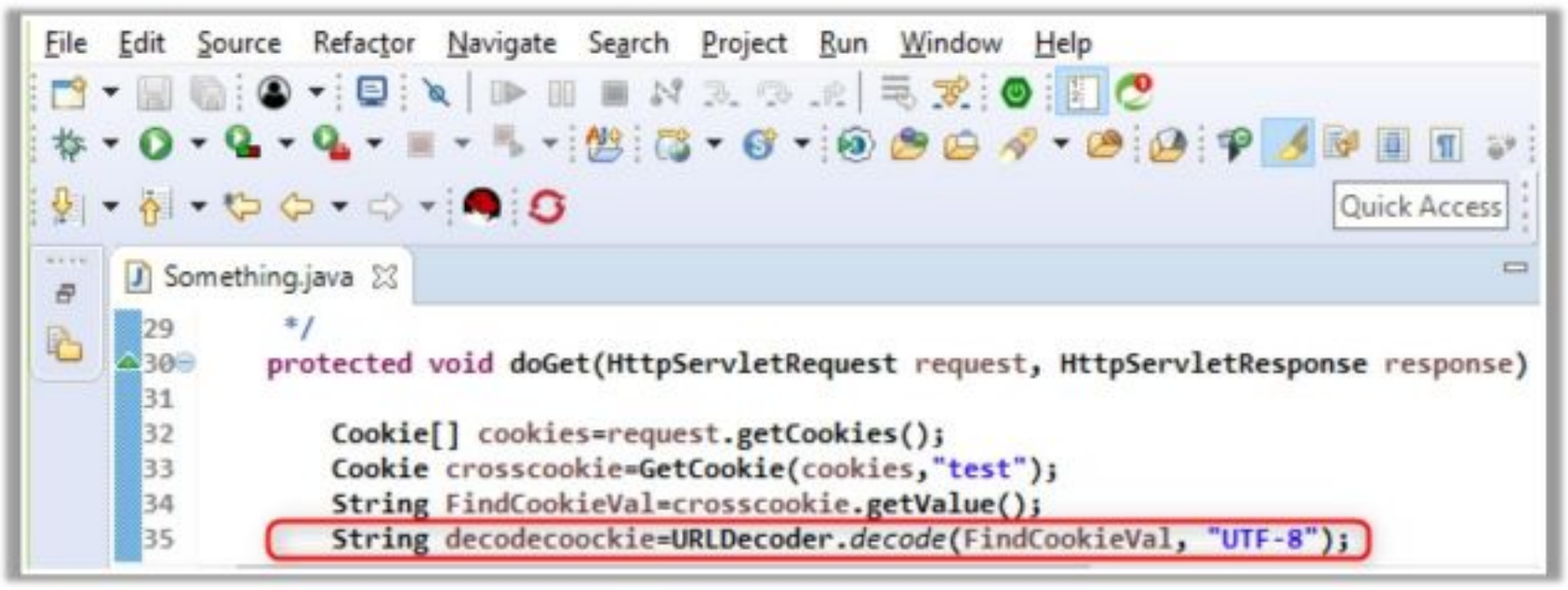


Recommendation: Always use a secure algorithm with sufficient key size for setting passwords

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for Unsafe Decoding

- **Code Pattern**: Look for code where the developer is attempting to **decode** cookie value, if cookies are not safely decoded, attacker can easily abuse the code



Recommendation: Always cross check the **cookie value** before use and confirm whether it is double encoded

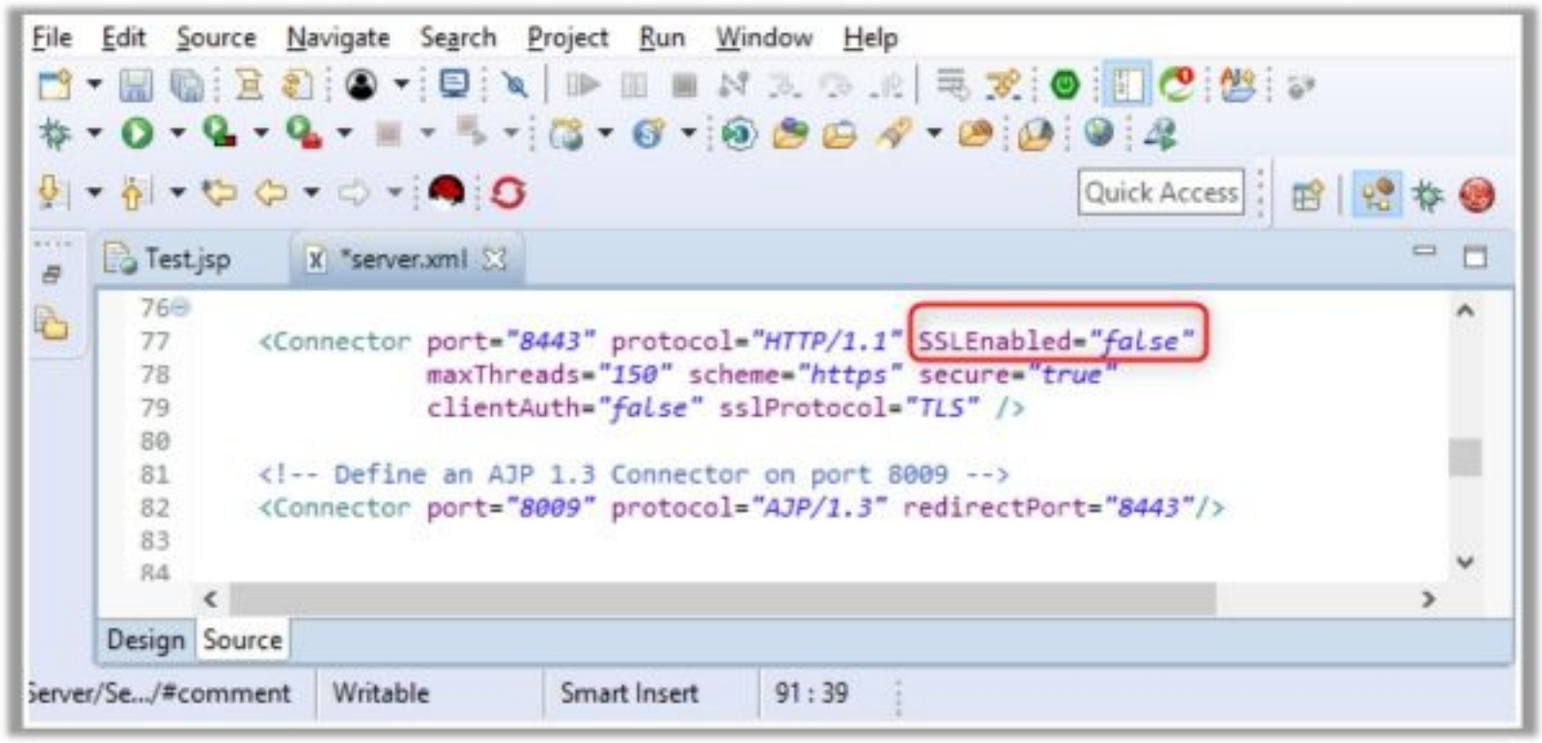
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Use of SSL

- **Code Pattern:** Look for configuration where the developer is attempting to set a **false** value to the **SSLEnabled** attribute in server.xml file

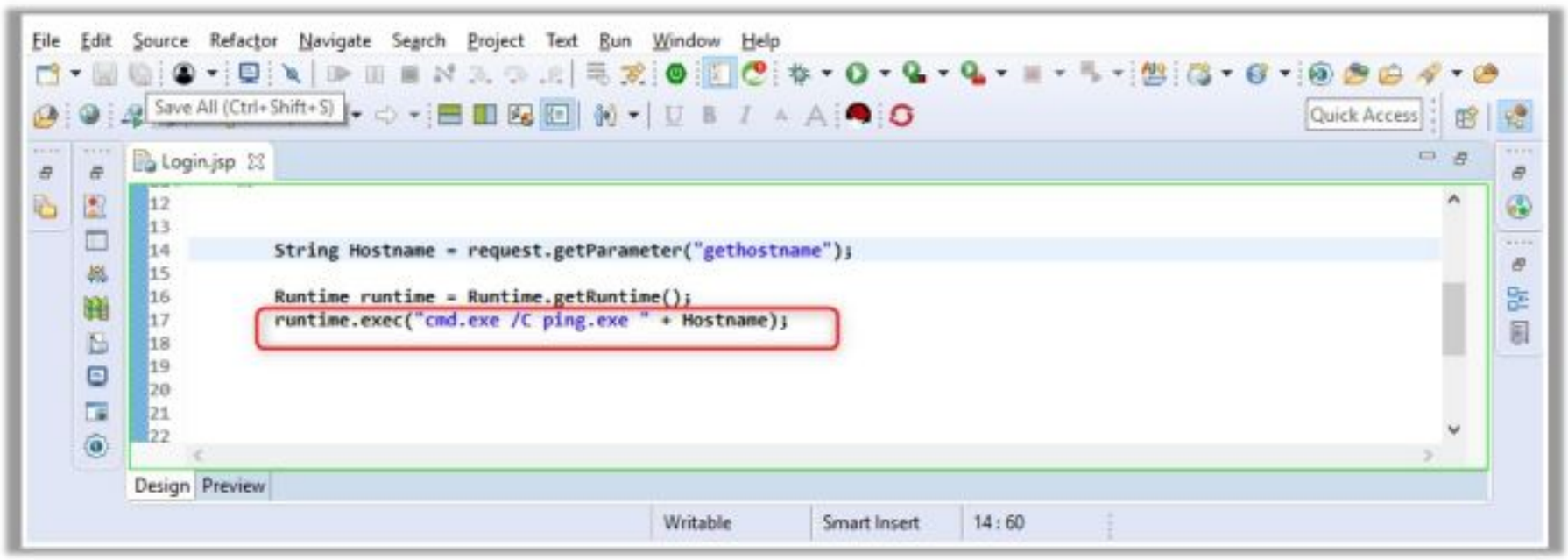


Recommendation: Always set the **SSLEnabled** attribute to **true**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for OS Command Injection

- **Code Pattern:** Look for code where the developer is allowing the **OS command** to execute from the application. Attackers can misuse this functionality and can execute any arbitrary command



Recommendation: Developers should avoid executing OS command using **runtime.exec.** Instead use **Java API's** to execute any OS related command

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for LDAP Injection

Code Pattern: Look for code where the developer allows the search filter without validation so attackers can input any malicious character and invoke the method. **Example**: **EmpInfo()**

```
File Edit Source Refactor Navigate Search Project Run Window Help
info.java
14
15 public class info {
16
17         private void EmpInfo(String EmploySN, String EmployPassword) throws NamingException {
18           Hashtable<String, String>  env = new Hashtable<String, String>();
19           env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
20           try {
21             DirContext dctx = new InitialDirContext(env);
22
23             SearchControls sc = new SearchControls();
24             String[] attributeFilter = {"cn", "mail"};
25             sc.setReturningAttributes(attributeFilter);
26             sc.setSearchScope(SearchControls.SUBTREE_SCOPE);
27             String base = "dc=example,dc=com";
28
29
30             String filter = "(&(sn=" + EmploySN + ")(EmployPassword=" + EmployPassword + "))";
31
32             NamingEnumeration<?> results = dctx.search(base, filter, sc);
33             while (results.hasMore()) {
34               SearchResult sr = (SearchResult) results.next();
35               Attributes attrs = (Attributes) sr.getAttributes();
36               Attribute attr = (Attribute) attrs.get("cn");
37               System.out.println(attr);
```

Recommendation: Always use a **whitelist** to validate user input so the user input contains only valid characters such as passwords containing only alphanumeric characters

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for XML Injection

Code Pattern: Look for code where the developer is attempting to use xml for writing data streams **write()** method which doesn't apply any **meta character encoding**

```
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
info.java  Employee.java
24    public void empInfo()
25    {
26
27        info emp=new info();
28
29  try {
30    Writer out = new StringWriter();
31    XMLStreamWriter writer = XMLOutputFactory.newInstance().createXMLStreamWriter(out);
32    writer.writeStartDocument();
33
34    writer.writeStartElement("Emp");
35    out.write("<FirstName>" + emp.fname+ "</FirstName>");
36    out.write("<LastName>" + emp.lname+ "</LastName>");
37
38    writer.flush();
39    out.flush();
40
41    writer.writeEndElement();
42    writer.flush();
43    }catch(Exception e) {
44
45    }
46    }
```

Recommendation: Always use **meta character encoding** while writing data into xml

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Unsafe use of request parameter to execute SQL Query

- **Code Pattern:** Look for code where the developer is attempting to execute a query directly received as **request parameter**. Attackers can manipulate these request parameters and execute a malicious query

```
18 <html>
19 <head>
20 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
21 <title>Insert title here</title>
22 </head>
23 <body>
24 <%
25    String sql=request.getParameter("query");
26    PreparedStatement ps=null,ps1=null;
27    Connection con1=null,con=null;
28    ResultSet rs=null,rs1=null;
29    int maxid=0;
30 Class.forName("oracle.jdbc.driver.OracleDriver");
31       con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","system","admin");
32       ps=con.prepareStatement(sql);
33
34       rs=ps.executeQuery();
35       while(rs.next())
36       {
37              maxid=rs.getInt(1)+1;
```

- **Recommendation:** Never use a **request parameter** for executing a query

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for Insecure File Upload

- **Code Pattern:** Look for code where the developer is attempting to upload a **file without proper validation**. An attacker can send any file and execute a command from the file

```
28 try{
29 Part filePart = request.getPart("file");
30 String fileName = filePart.getName();
31 boolean fileUploaded = false;
32
33 if(filePart != null && fileName != null){
34
35 InputStream fileContent = filePart.getInputStream();
36 byte[] bytes = IOUtils.toByteArray(fileContent);
37
38
39 OutputStream out = new FileOutputStream(UploadPath + fileName);
40
41 out.write(bytes);
42 fileUploaded = true;
43
44 fileContent.close();
45 out.flush();
46 out.close();
47 }}
```

- **Recommendation**: Always **validate file extensions** while uploading files

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Directory Traversal

- **Code Pattern:** Look for code where the developer is attempting to read a file if the file exists. Attackers can manipulate path variables and can access **restricted files** and **directories**

```
27
28          String fileName = request.getParameter("file");
29
30          if (fileName == null) {
31              return;
32          }
33
34          File downloadedFile = new File(UploadPath + fileName);
35
36          if (!downloadedFile.exists()) {
37              return;
38          }
39          OutputStream out1 = null;
40          FileInputStream in = null;
41          try {
42              out1 = response.getOutputStream();
43              in = new FileInputStream(downloadedFile);
44              byte[] buffer = new byte[4096];
45              int length;
46              while ((length = in.read(buffer)) > 0) {
47                  out1.write(buffer, 0, length);
48              }
```

Recommendation: Always **validate** the **location** of a file and read file from that location

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for Sensitive Information Exposure

- **Code Pattern:** Look for code where the developer is attempting to store sensitive information into a **cookie.** A cookie can be read by attackers

```
import java.io.IOException;
public class ValidateUser {


public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException  {

String currentDir = System.getProperty("user.dir");
Cookie cookie=new Cookie("cwd", currentDir);

res.addCookie(cookie);
```

Recommendation: Do not store **sensitive information** in **cookies**

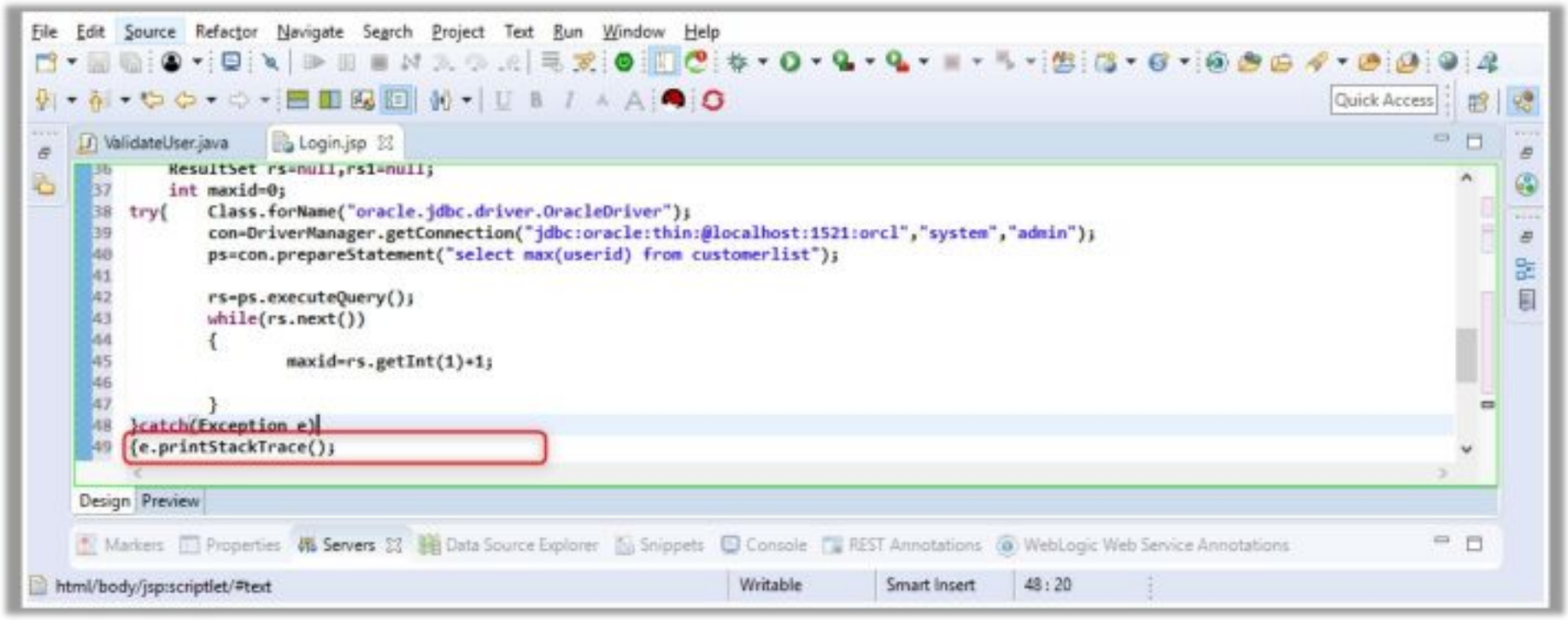
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Sensitive Information Leakage

- **Code Pattern:** Look for code where the developer is using **printStackTrace** while handling exceptions. **printStackTrace** gives detailed information about the exception occurred

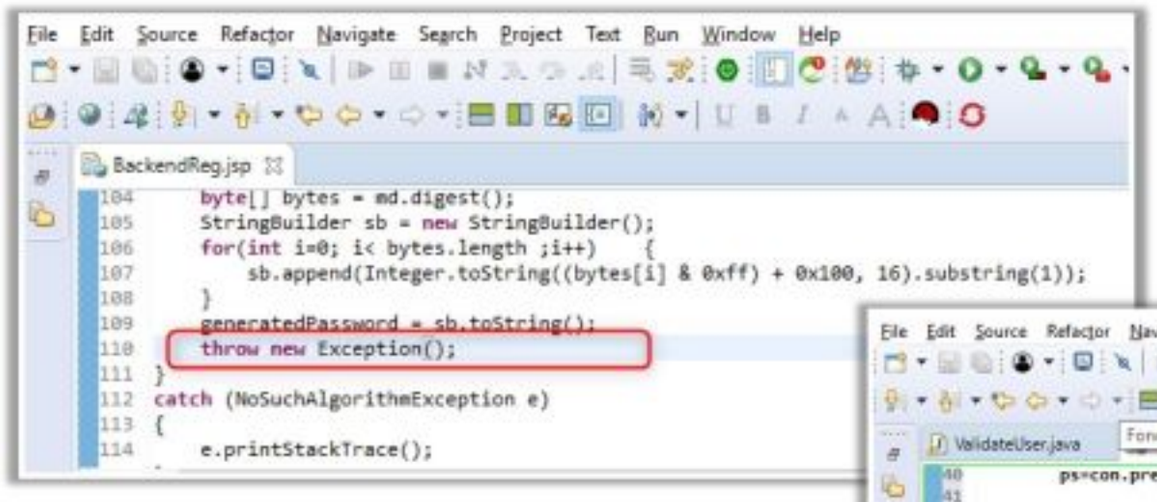


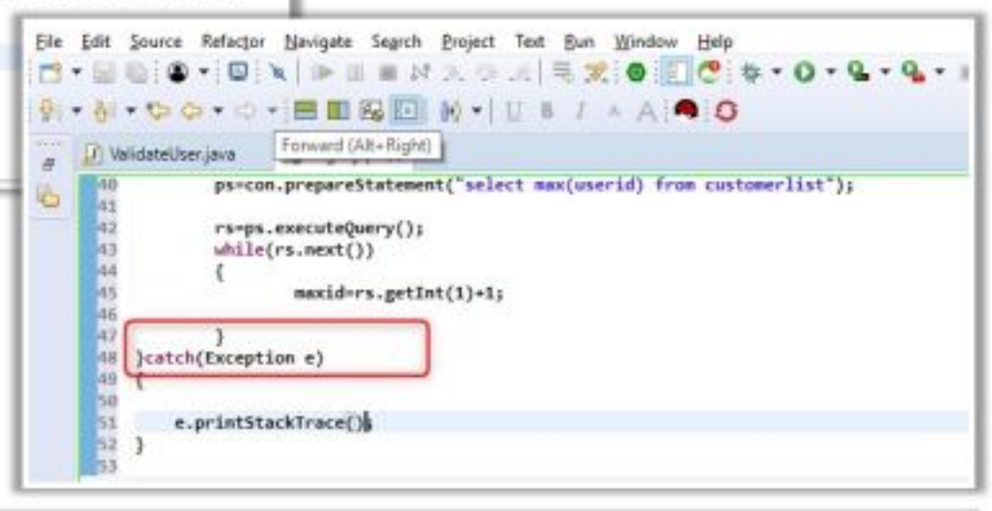
Recommendation: Do not print the **stack trace** when the exception is caught

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for Generic Exception Throwing and Catching

- **Code Pattern:** Look for code where the developer is attempting to **throw** and **catch** an exception with **super class**





Recommendation : Always **throw** and **catch** exceptions with a specific type of **exception class**

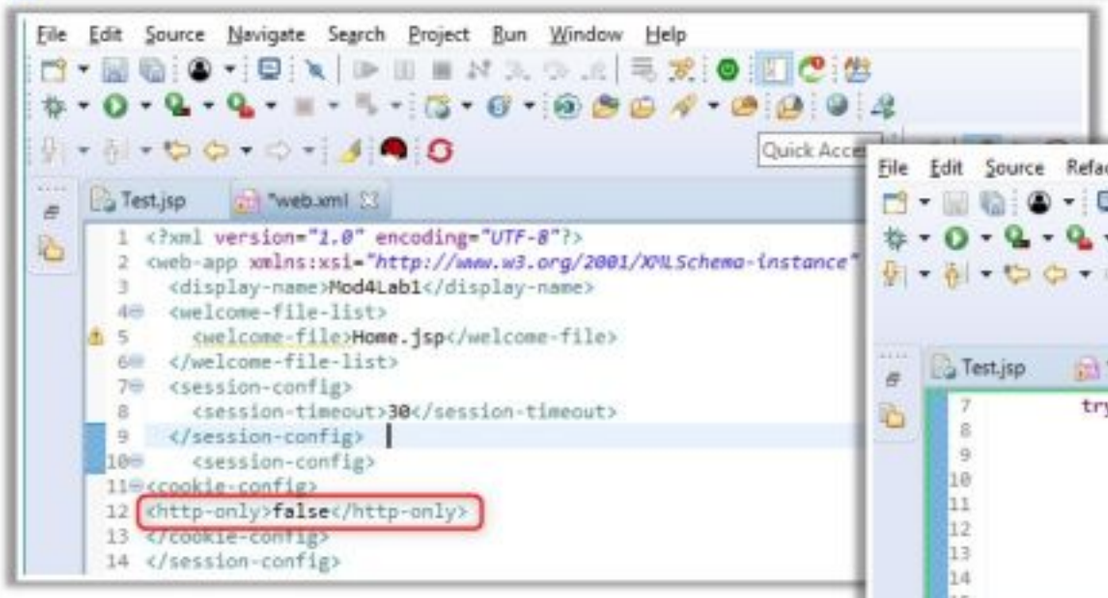
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Cookies Vulnerable to Client-side Scripts Attacks

- **Code Pattern:** Look for the configuration where the developer is attempting to set the **httpOnly** attribute to **false**

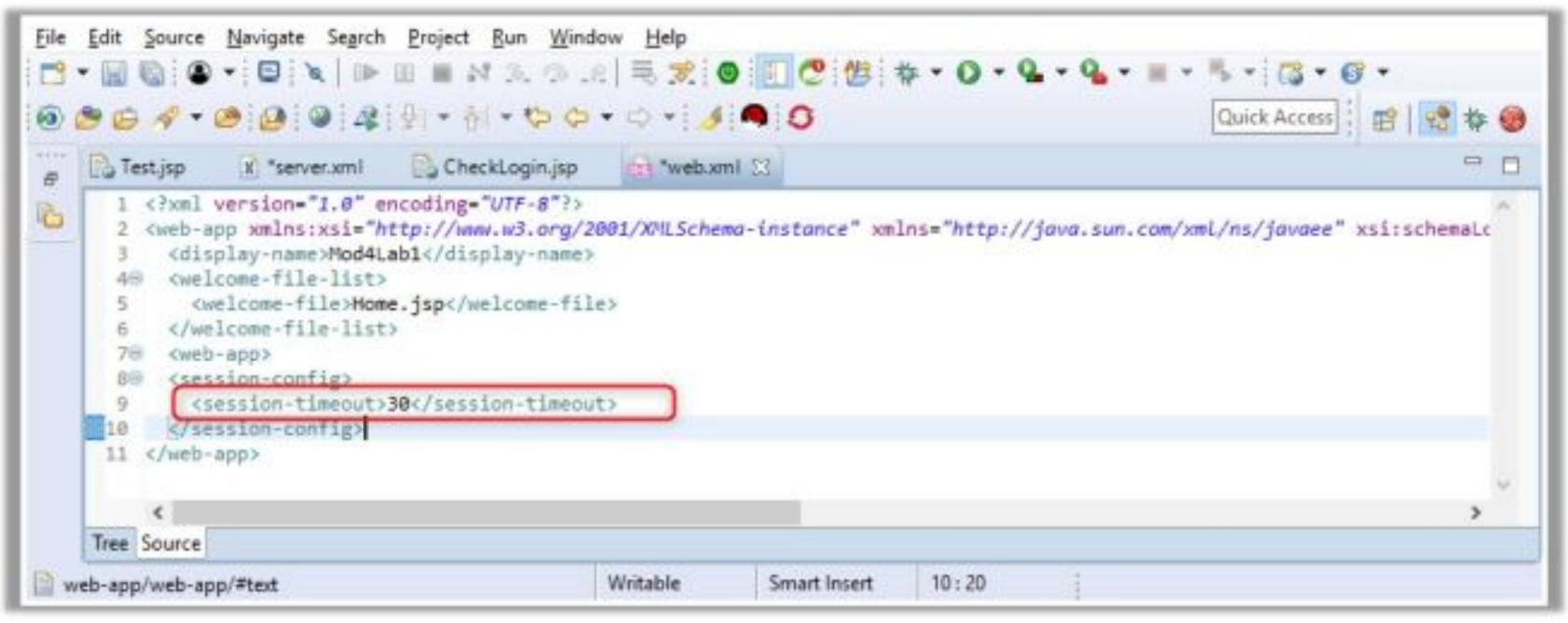


Recommendation: Always set **httpOnly** attribute to **true**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review for Cookies Vulnerable to CSRF Attacks

- **Code Pattern:** Look for configuration where the developer is attempting to set the **session-timeout** value to a large value in the **web.xml** file



Recommendation: Always set **session-timeout** value to as minimum as possible

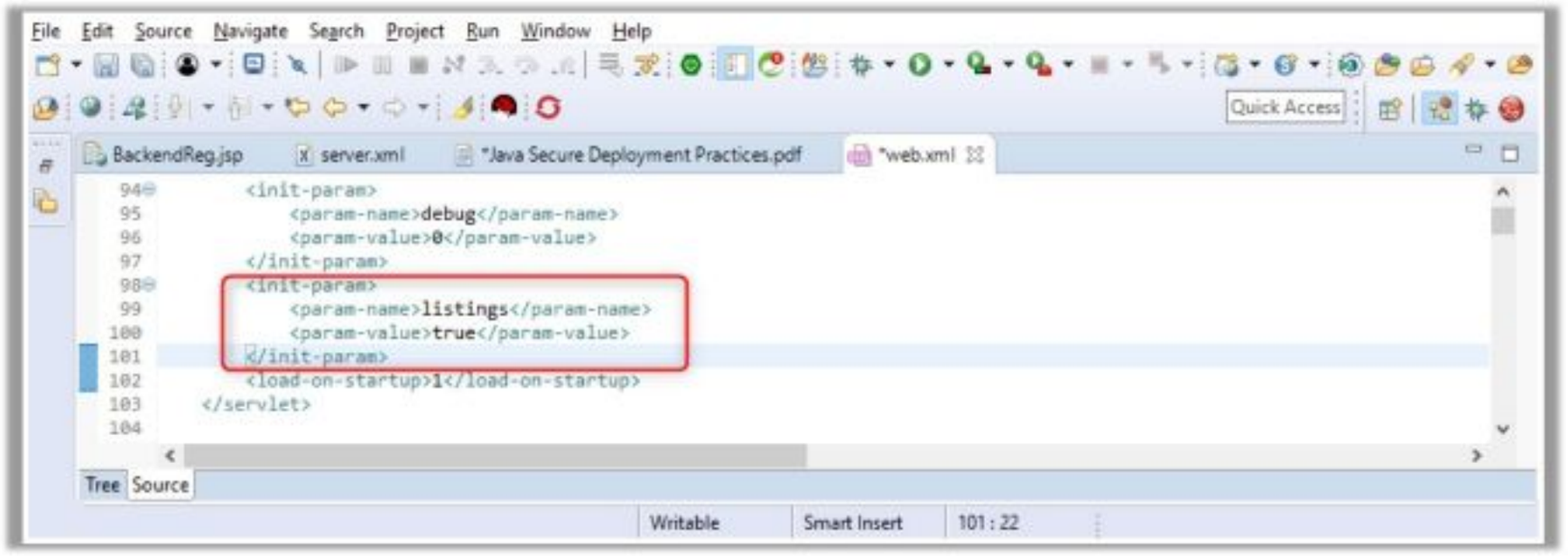
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Code Review for Enabling Directory Listing

- **Code Pattern:** Look for configuration where the developer is attempting to set the directory **listing parameter** value to **true** in web.xml file of the tomcat server



- **Recommendation:** Do not enable the directory listing feature. Set **param** value as **"false"** in web.xml

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Code Review: Check List Approach

What does a Checklist Contain?

- Data Validation and Encoding Controls
- Encryption Controls
- Authentication and Authorization Controls
- Session Management
- Exception Handling
- Auditing and Logging
- Security Configurations

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Sample Checklist: Input Validation

- ✔ Ensure that all external inputs are validated
- ✔ Ensure that data is validated server-side (security should not rely on client-side validations)
- ✔ Ensure that the application validates respective data types and rejects unexpected input
- ✔ Ensure that the application efficiently evaluates input length
- ✔ Ensure that there is strong separation between data and commands
- ✔ Ensure that there is strong separation between data and client side scripts
- ✔ Ensure that data is validated for special characters before being passed to SQL, LDAP, OS and third party commands
- ✔ Ensure that http headers are validated for each request (e.g. referer)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Sample Checklist: Authentication

- Ensure that user credentials are encrypted in the data store
- Ensure that security policies are configurable (not hardcoded)
- Ensure that standard security frameworks are used (instead of custom code)
- Ensure that SSL is used to protect user credentials and authentication tokens
- Ensure that authentication cookies are not persisted
- Ensure that authentication cookies are encrypted
- Ensure that cookie names and paths are used
- Ensure that application handles user management events such as authentication failure, password reset, password change, account lockout and cancel account
- Ensure that application handles suspicious events such as multiple failed logon attempts, session replay and attempted access to restricted resources
- Ensure that strong passwords policies are enforced
- Ensure that authentication credentials are not passed by HTTP GET

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Sample Checklist: Authorization

Ensure that authentication and authorization should be the first logic executed for each request
Ensure that authorization checks are granular (page and directory level)
Ensure that application denies access to pages and data by default
Ensure that application re-authenticates for requests that have side-effects
Ensure that ACLs are configured for all files
Ensure that authorization is performed based on roles defined
Ensure that application implements strong authorization and can not be bypassed through parameter manipulation
Ensure that authorization cannot be bypassed through cookie manipulation

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Sample Checklist: Session Management

Ensure that application should not pass session parameters in URLs
Ensure that duration for session cookies expiration should be as minimum as possible
Ensure that cookies are encrypted
Ensure that session data is validated
Ensure that private data in cookies is kept to a minimum
Ensure that application avoids excessive cookie use
Ensure that session id is complex and generated randomly
Ensure that session storage is secure
Ensure that application properly handles invalid session ids
Ensure that session timeout are enforced
Ensure that logout invalids the session
Ensure that session resources are released when session invalidated

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Sample Checklist: Cryptography

- Ensure that sensitive data is secured in memory, storage and transit
- Ensure that sensitive information is not passed to/from non-SSL pages
- Ensure that proper SSL is set up
- Ensure that SSL provider supports only strong algorithms
- Ensure that decryption services are protected by authentication/authorization
- Ensure that application securely stores cryptographic keys

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Sample Checklist: Exception Handling

- Ensure that the application fails securely
- Ensure that error messages do not reveal sensitive information
- Ensure that system errors are never shown to users
- Ensure that resources are released and transactions are rolled back when there is an error

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Sample Checklist: Logging

- Ensure that all user / system actions are logged
- Ensure that sensitive information is not logged (e.g. passwords)
- Ensure that Logging is enabled for user management events (e.g. password reset)
- Ensure that unusual activity such as multiple login attempts are logged
- Ensure that logs have enough detail to reconstruct events for audit purposes
- Ensure that logging is highly configurable (logging levels)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SAST Finding

- The SAST findings keep the development team informed about the findings gathered by the security team during the secure code review
- The findings generally include vulnerabilities, architectural problems, organization issues and any mismatch in the best practices followed
- The security team members may include the following features in the findings report:
  - **Select a meaningful title for the findings report**: The title should be clear and should convey the details of the issue clearly to the development
  - **Determine clearly the location of the issue**: The development team should easily identify the location of the issue from the report
  - **Provide detailed description of the security issues**: The security team should include the following details regarding the security issue:
    - A clear description about the vulnerability
    - Analysis of the possible attack areas
    - Description about the factors leading to the attack
- **Provide a description about the risk ratings**: The risk rating enables the development team regarding the value assigned to each security issue found. The report may also contain the explanation behind assigning any particular value to an identified security issue that may enable the development team to understand the impact of the security issue at a glance. The commonly used risk ratings during code review are as follows:
  - Critical
  - High
  - Moderate
  - Low
- The security team should follow some standards while assigning risk ratings to the security issues during secure code review
- **Provide proper corrective measures to the development team**: The security team after the secure code review may provide certain corrective measures that enable the development team to easily rectify the security issues. The rectification measures may also include:
  - Other course of action
  - Total amount of effort required for rectification

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

SAST Report

- **The SAST report consists of the following details:**

1. A **list of vulnerabilities** and **its related details** along with the severity level
2. The **rectification measures** taken for controlling the vulnerabilities and errors found in the code
3. **Graphical representation** of the **security measures** required for the application
4. **Control measures** that can be taken for future security as well

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SAST Reporting

- Report the findings of the code review in a structured way
- The report should include:
  - **Weakness Metadata**
    - Details of Place to find the issue
  - **Through Description**
    - Detailed description of the issue
  - **Recommendation**
    - Recommendation to mitigate the issue
  - **Assign appropriate Priority**
    - Give priority to the issue for the developer to prepare his to –do-list

SAMPLE REPORT

SQL INJECTION

Location: \myProject\UserModule\User.jsp?username=

Description: The code below builds a dynamic SQL statement without validating the input from request parameter. This may lead to SQL INJECTION

```
String query="select FirstName,LastName from Users
WHERE UserName='" +request.getParameter("username")
+ "'"; PreparedStatement
stmt=con.prepareStatement(query);
ResultSet rs=stmt.executeQuery();
```

Recommendation: Use parameterized SQl query; do not dynamically concatenate the request parameter value to the string query.

Ref:https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

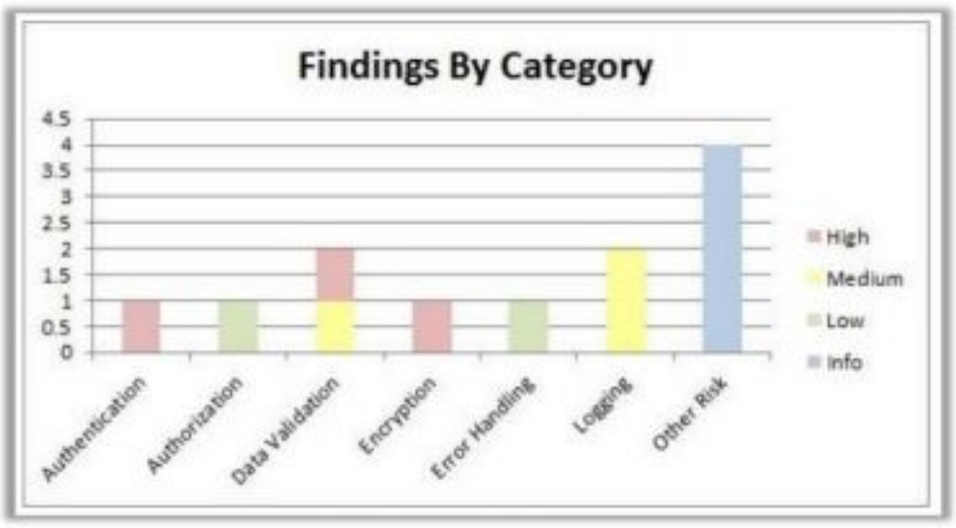
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

SAST Reporting (Cont'd)

Review /Engagement Reference:			
Package/Component/Class Name/Line Number:			
Finding Title:			
Severity: High			
Finding Description	Location(s)	Risk Description	Recommendation
No input validation of the **HTTPRequest object.getID()** function. Lack of input validation may make the application vulnerable to many types of injection	com.inc.dostuff.java Lines 20, 55,106 com.inc.main.java Lines 34, 99	Discussion of the likelihood and impact to the business if the flaw were to be exploited.	It is critical that this be addressed prior to deployment to production

Finding Report Sample



Risk Rating

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Dynamic Application Security Testing

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Dynamic Application Security Testing (DAST)

- DAST is a security testing technique which involves **simulating** attacks against the application and analyzes how the application behaves
- The application is tested **dynamically** from the outside when application is running
- DAST is generally performed by the penetration testers or security practitioners on a working system in pre-production, a test environment, or even in production
- These individuals typically uses automated web application vulnerability scanners to conduct DAST
- These scanners believe in **fuzzing** the application inputs with attack payloads in order to detect security weaknesses
- Typical web application vulnerability scanners are classified into:
  - **Classic application security scanners:** They crawl and automatically scan all detected webpages. However, they have limited coverage within the application as they do not have understanding of business flows
  - **Proxy-based security testing tools**: These are scanners integrated in web application proxies which provide fine-grained control over each request that gets scanned

DAST (Dynamic)



Scans interface of running application

- Black box testing
- Simulates a live attacker
- Sends HTTP requests
- Analyses HTTP responses

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Types of DAST

Types of DAST

Automated Application Vulnerability Scanning

- Security tester uses classic **application security scanners** to scan the web application for vulnerabilities

Manual Application Security Testing

- Security tester uses **Proxy-based** security testing tools to craft and send request manually and analyze the responses from the application

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

SAST vs DAST

SAST	DAST
White box security testing	Black box security testing
Requires a source code	Requires a running application
Finds vulnerability earlier in SDLC	Finds the vulnerability towards the end of SDLC
Less expensive to fix vulnerability	More expensive to fix vulnerability
Runtime and environment related issues can't be discovered	Runtime and environment related issues can be discovered
Typically supports all kinds of software	Typically scans only apps like web application and web services

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Automated Application Vulnerability Scanning Tools

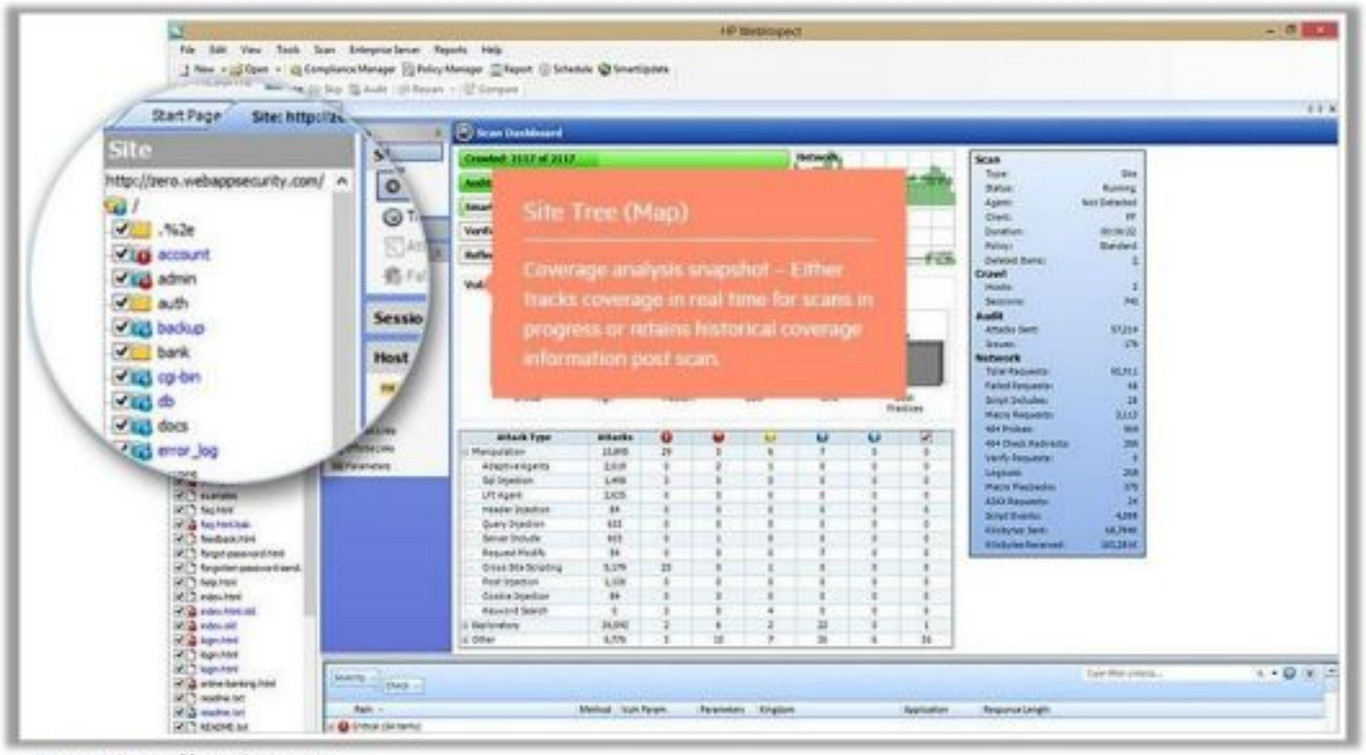
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Web Application Security Scanners: WebInspect

- WebInspect is a web application security assessment solution designed to thoroughly analyze today's complex **web applications** and **web services** for security vulnerabilities
- It provides the broadest dynamic application security testing (DAST) coverage and detects new types of vulnerabilities that often go undetected by **black-box security testing** technologies

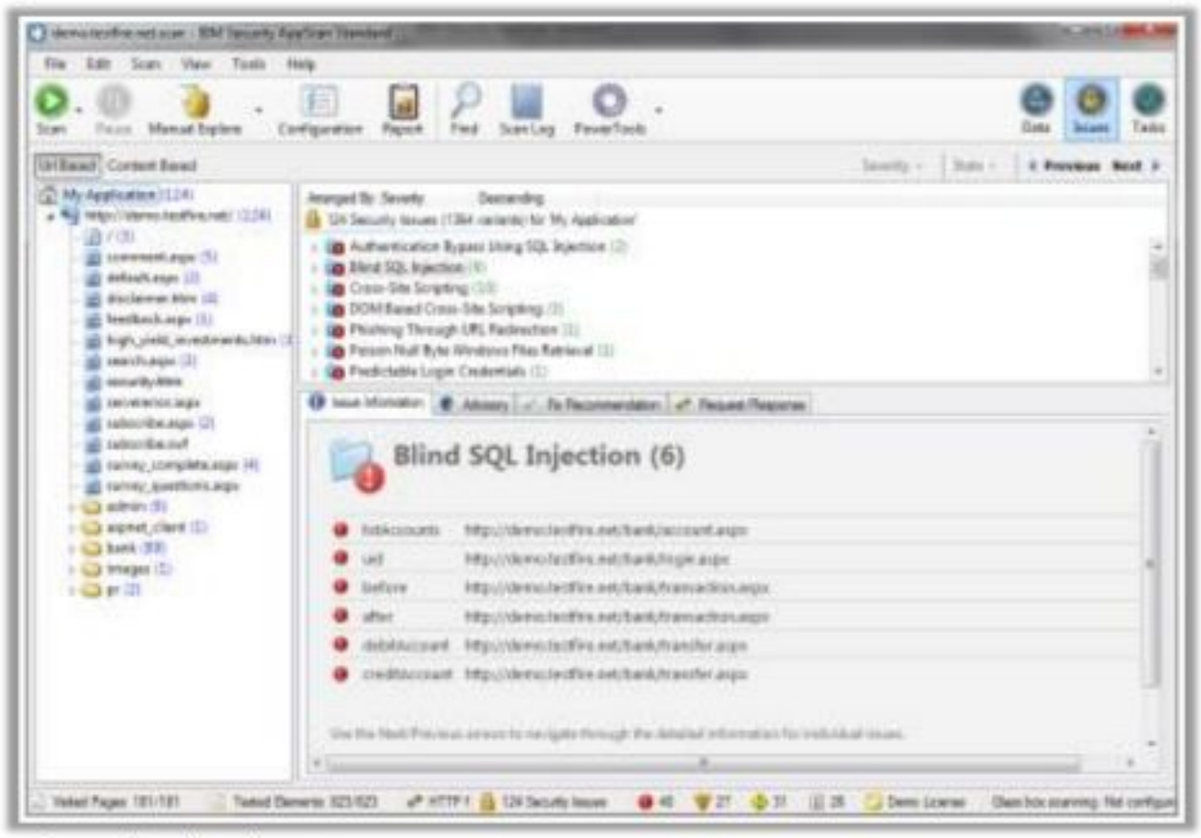


Source: *https://saas.hpe.com*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Web Application Security Scanners: IBM SecurityAppScan

- **IBM** Security **AppScan** helps organizations decrease the likelihood of web application attacks and costly data breaches by automating application security vulnerability testing



Source: *https://www.ibm.com*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Additional Web Application Vulnerability Scanners

Acunetix WVS
https://www.acunetix.com

AppScan
https://www.ibm.com

Websecurify
https://www.websecurify.com

N-Stalker
https://www.nstalker.com

Nessus
https://www.tenable.com

NetSparker
https://www.netsparker.com

NeXpose
https://www.rapid7.com

WebKing
https://www.parasoft.com

ParosPro
https:// www.milescan.com

Retina Web Security Scanner
https://www.beyondtrust.com

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Proxy-based Security Testing Tools

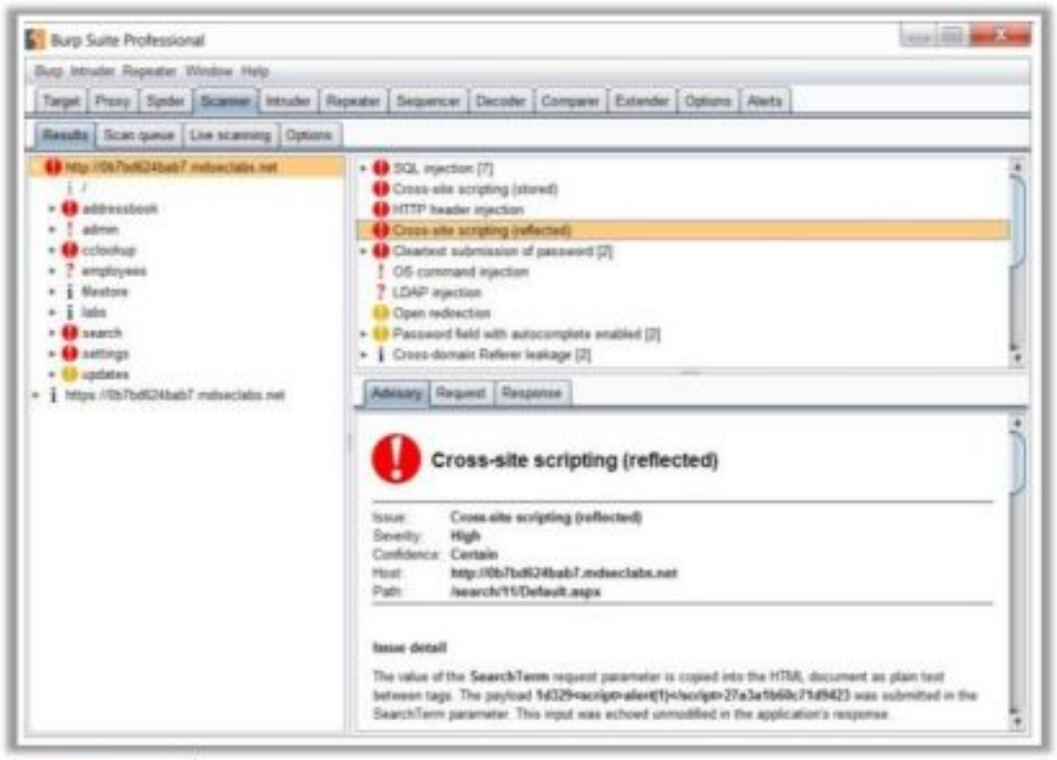
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Burp Suite

- Burp Suite is an integrated platform for performing security testing of web applications. Its various tools work seamlessly together to support the entire testing process, from initial mapping and analysis of an **application's attack surface**, through to finding and exploiting security vulnerabilities
- Install and configure Burp Suite in the browser of your interest



Source: *https://portswigger.net*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

OWASP Zed Attack Proxy (ZAP)

- **OWASP ZAP** is an open source, easy to use, integrated penetration testing tool for finding vulnerabilities in web applications

DAST using OWASP ZAP

1. Install and configure ZAP in the browser of your interest
2. Insert the URL of web application that you want to test and start the attack



Source: *https://www.owasp.org*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Additional Proxy-based Security Testing Tools

Vega
https://www.acunetix.com

Wfuzz
http://www.edge-security.com

Paros proxy
http://www.parosproxy.org

Ratproxy
https://code.google.com

Firefox Tamper Data
https://addons.mozilla.org

webstretch
https://sourceforge.net

WebScarab
https://www.owasp.org

Wapiti
http://wapiti.sourceforge.net

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Choosing Between SAST and DAST

1. **SAST** is **white box** security testing approach while **DAST** is a **black box** security testing approach
2. These approaches work differently on their part to **find security flaws** in the application
3. **Relying on DAST** alone does not ensure that security flaws will **exist** that you have not tested for
4. Relying on SAST alone does not ensure that security flaw will not exist that may arise when application is running
5. After SAST, security tester may have a better chance of **finding different problems** when application is running
6. Thus, both **SAST** and **DAST** are required in a secure software development program

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Module Summary

- SAST is one of the software security assurance approaches executed in an attempt to identify security-related weaknesses in the code
- Manual Source Code Review involves manually inspecting the source code line by line to detect any defects and security related flaws
- The static code analysis tools automate the entire process of secure code review; however, they scan the code superficially
- With manual secure code review, you can analyze and detect many security vulnerabilities especially those missed out during the static code analysis
- DAST is a security testing technique which involves simulating attacks against the application and analyzing how the application behaves
- SAST is a white box security testing approach while DAST is a black box security testing approach

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

JAVA

C | ASE

Certified | Application Security Engineer



Module 10

Secure Deployment and Maintenance

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

Module Objectives

1. Understand the Importance of Secure Deployment
2. Discuss Security Practices at Host Level
3. Discuss Security Practices at Network Level
4. Discuss Security Practices at Application Level
5. Discuss Security Practices at Web Container Level (Tomcat)
6. Discuss Security Practices at Oracle Database Level
7. Discuss Security Maintenance and Monitoring Activities

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Secure Deployment

1. Deployment is the **last phase** of software development lifecycle where the application is moved from development environment to production environment
2. It is the first time where the **system administrator** is involved in software development
3. The application can be exposed to different types of attacks, if an administrator commits **configuration mistakes** in the deployment phase
4. **Security** should also be the key consideration while deploying any application

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Prior Deployment Activity: Check the Integrity of Application Package before Deployment

- Software vendors generally provide **checksum** of the application with the application package
- System administrators should again generate a checksum of the application and compare it with the checksum provided with the package to ensure the integrity of the application
- Two **identical checksums** confirm that the application is not tampered during the delivery

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Prior Deployment Activity: Review the Deployment Guide Provided by the Software Vendor

- Use **deployment checklists** provided by the software vendor
- **Do not build and use** your own deployment checklist as it may affect the application security

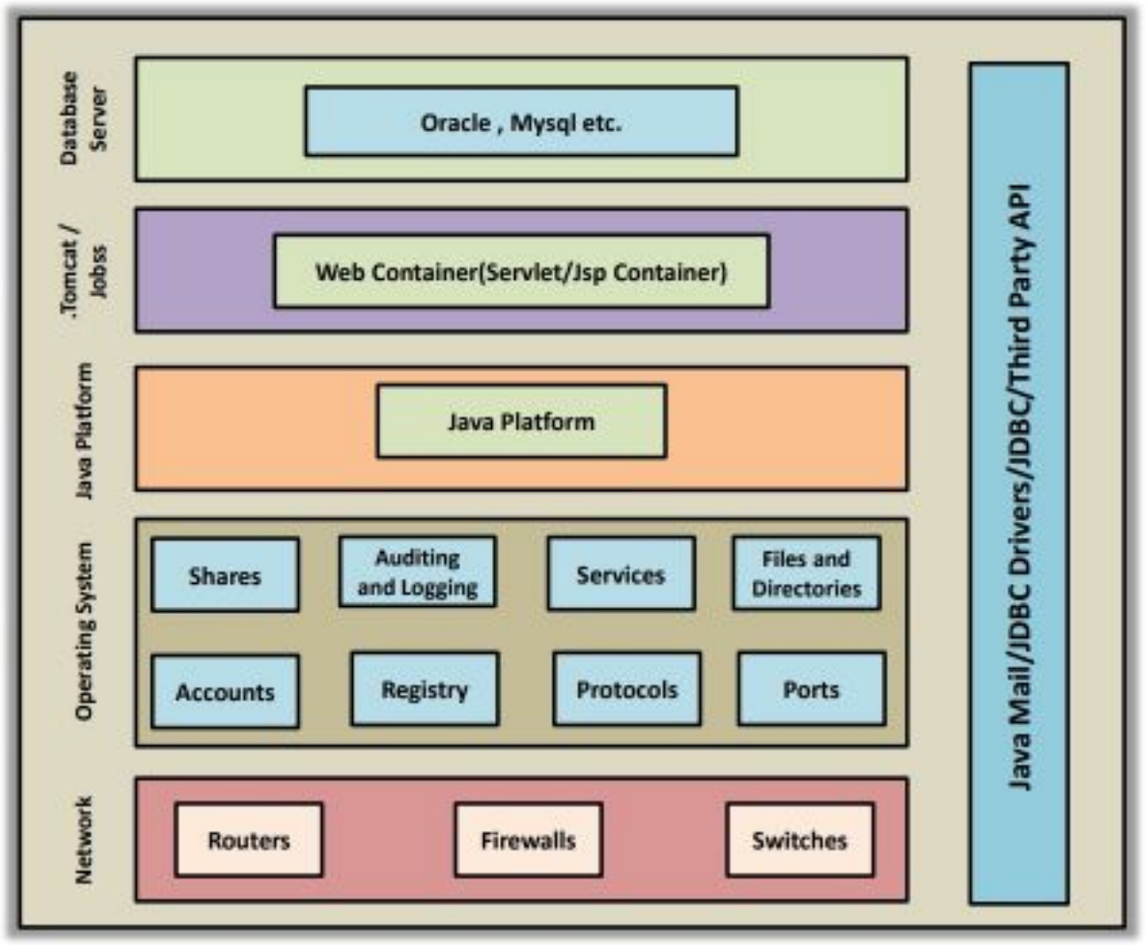
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Deployment Activities: Ensuring Security at Various Levels

- Secure deployment involves ensuring security at various levels from bottom to top
- It involves considering security related to the potential customer environments which include:
  - **Host Level** Deployment Security
  - **Container Level (Tomcat)** Deployment Security
  - **Oracle Level** Deployment Security



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Ensuring Security at Host Level

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Check and Configure the Security of Machine Hosting Web Server, Application Server, Database Server and Network Devices

I. Firstly, the production environment will be secured, if the machine hosting the **web server** is secured

II. The physical security of a host machine, its OS security, and the security of the all other software installed on the machine should be ensured

III. Check the **host server** and all of the services upon which the application security will depend

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Physical Security

- Ensure that **server machine is placed** in a secured area
- Ensure that an appropriate physical access control is implemented to restrict unauthorized physical access



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Host Level Security

Component	Characteristics
Patches and Updates	Not deploying patches and updates to the web server can allow attackers to run malicious code or intrude the server
Services	Services are easy targets for attackers to access the privilege and exploit the functioning of the server Do not run the service that is not required for the server functioning It is important to monitor running services
Protocols	Do not implement protocols that are insecure and can not provide encrypted authentication Example: Telnet, File Transfer Protocol(FTP), Simple Mail transfer Protocol(SMTP)
Accounts	Audit the user accounts regularly to avoid any malicious activities Provide limited privileges to account to prevent privilege elevation Delete accounts that are no more beneficial for the application Implement strong password policies
Files and Directories	Implement NTFS permissions on files and directories on web server this will allow access to only Windows services
Shares	Secure file shares process with NTFS permissions Limited shares reduces intrusion possibility on web server
Ports	Services running on server listen to specific ports Auditing ports regularly will help to detect any irregular service on the server
Registry	Secure the registry by: Applying restricted Windows ACL Blocking remote registry administration

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Ensuring Security at Network Level

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Network Level Security: Router

Component	Characteristics
Patches and Updates	Router operating system is patched with up-to-date software
Protocols	Unused protocols and ports are blocked Ingress and egress filtering is implemented ICMP traffic is screened from the internal network TTL expired messages with values of 1 or 0 are blocked (route tracing is disabled) Directed broadcast traffic is not forwarded Large ping packets are screened Routing Information Protocol (RIP) packets, if used, are blocked at the outermost router
Administrative access	Unused management interfaces on the router are disabled A strong administration password policy is enforced Static routing is used Web-facing administration is disabled
Services	Unused services are disabled (for example bootps and finger)
Auditing and logging	Logging is enabled for all denied traffic Logs are centrally stored and secured Auditing against the logs for unusual patterns is in place
Intrusion detection	IDS is in place to identify and notify an active attack

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Network Level Security: Firewall

Component	Characteristics
Patches and Updates	Firewall software and OS are patched with latest security updates
Filters	Packet filtering policy blocks all but required traffic in both directions Application-specific filters are in place to restrict unnecessary traffic
Logging and Auditing	All permitted traffic is logged Denied traffic is logged Logs are cycled with a frequency that allows quick data analysis All devices on the network are synchronized to a common time source
Perimeter networks	Perimeter network is in place if multiple networks require access to the server Firewall is placed between untrusted networks

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Network Level Security: Switch

Component	Characteristics
Patches and Updates	Latest security patches are tested and installed, or the threat from known vulnerabilities is mitigated
VLANs	Make sure VLANs are not overused or overly trusted
Insecure defaults	All factory passwords are changed Minimal administrative interfaces are available Access controls are configured to secure SNMP community strings
Services	Unused services are disabled
Encryption	Switched traffic is encrypted

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Ensuring Security at Application Level

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

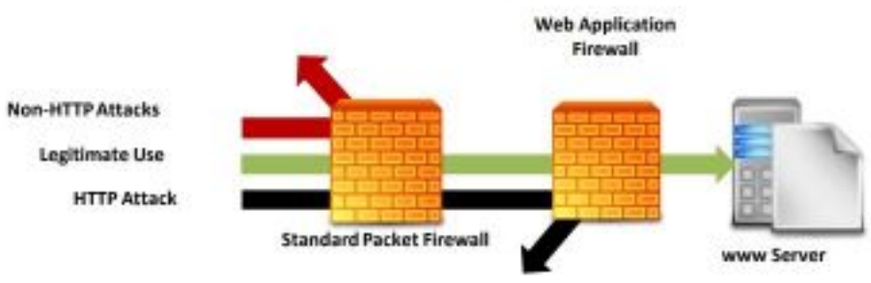
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

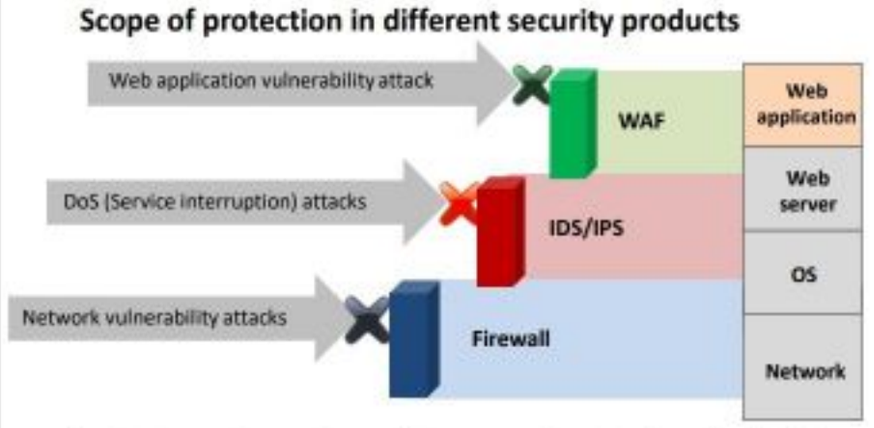
Web Application Firewall (WAF)

1. A traditional firewall cannot secure webservers from a malicious traffic attack as the attack occurs at layer 7 of the network stack
2. A **WAF** provides a security layer that protects the **web server** from the malicious traffic
3. WAF is either **appliance based** or **cloud based** and gets deployed through a proxy placed ahead of the web application
4. It uses **a rule based filter** that monitors and analyzes the traffic before it reaches the web application

Placement of WAF and its Working



Scope of protection in different security products



Web application attack cannot be completely prevented by existing firewall and IDS/IPS

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Benefits of WAF

- WAF implementation secures the **existing and productive web applications**
- Many WAF's have functionalities that can be used in design process and minimizing the work load
- WAF provides **cookies protection** with encryption and signature methodology
- It secures applications from **cross-site request forgery** and negates **parameter tampering** by **URL encryption**
- WAF can detect **data validation issues** by in-depth testing of character, character length, the range of a value, etc.

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

WAF Limitations

- Do not consider WAF as a replacement for a proper **application security** such as user authentication or input filtering
- WAF is not a technology which if once deployed, can be ignored by the administrator
- The working of WAF is different from the **next-generation firewall (NGFW).** WAF inspects traffic based on a particular protocol unlike NGFW which can make changes in the existing network
- WAF does not provide complete security from all web attacks as it can not read database commands
- Only if WAF manages the session itself, it can partially prevent issues like session fixation and anti-automation
- Deployment of WAF does not ensure security from **false positives**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

WAF Vendors



Barracuda WAF

Source: *http://www.barracuda.com*



A10 WAF

Source: *https://www.a10networks.com*



Citrix NetScaler MPX WAF

Source: *https://www.citrix.co.in*



Imperva WAF

Source: *https://www.imperva.com*



MonitorApp WAF

Source: *http://www.monitorapp.com*



Trustwave WAF

Source: *https://www.trustwave.com*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

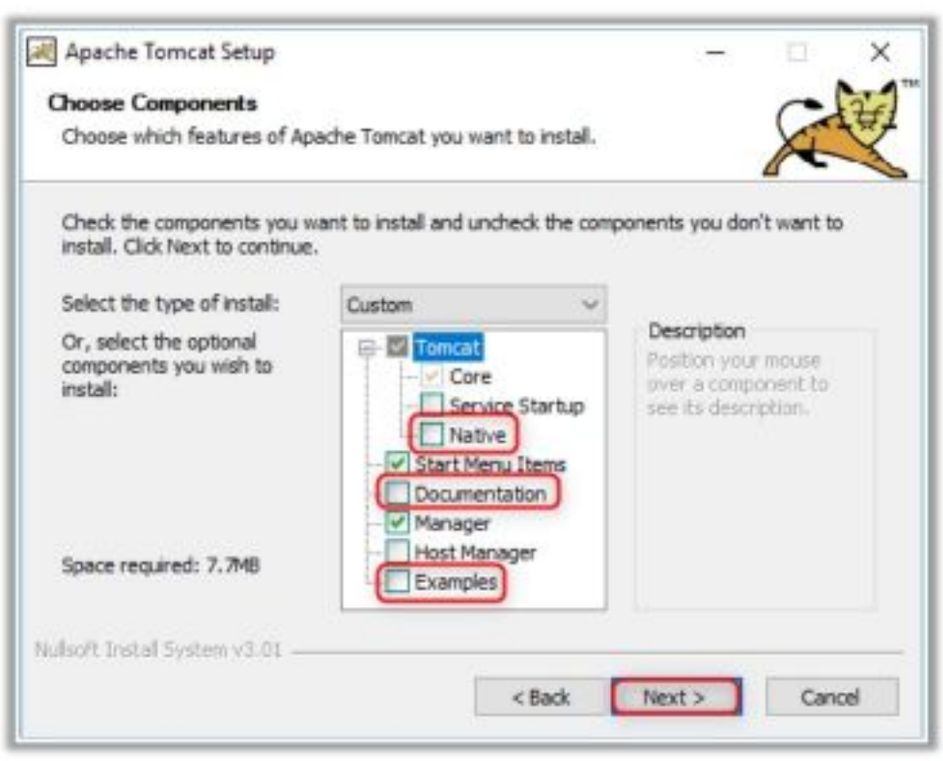
@ErrorCatchers

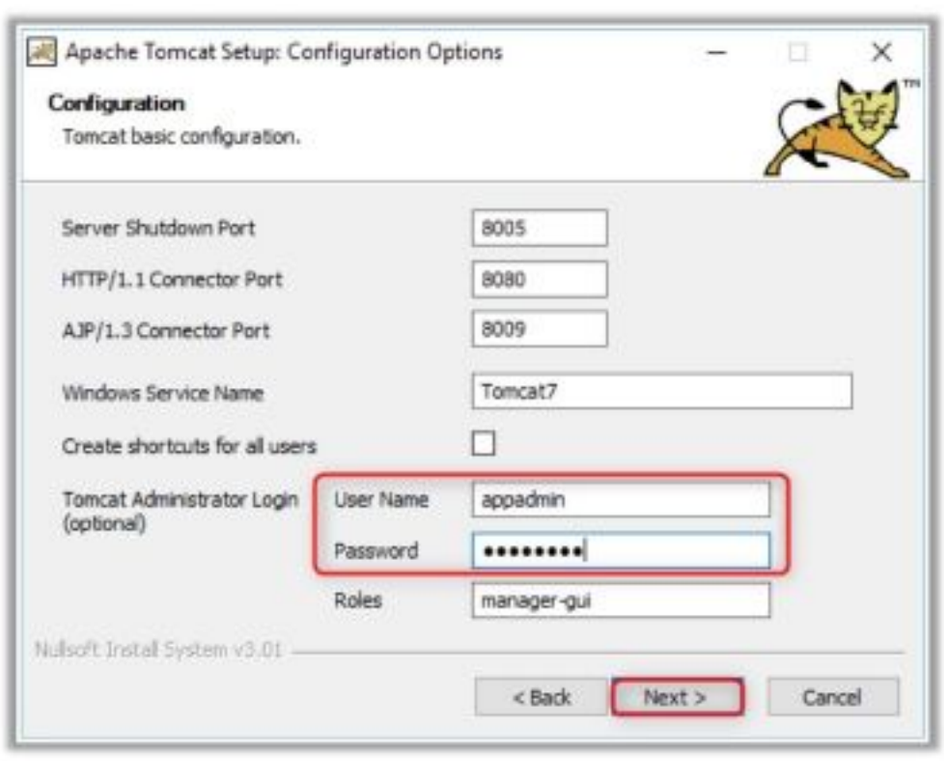
Ensuring Security at Web Container Level (Tomcat)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Install and Configure Tomcat Securely

- While installing **Tomcat** uncheck the options **Native, Documentation,** and **Examples** in **Choose Components** window
- Select **User Name** other than **admin**, and give a secure **Password** as per the organization password policy in **Configuration** window





Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Remove Server Banner

- The **server banner** gives out the product and version details which leads to information disclosure vulnerability
- To prevent Tomcat from displaying product and version details, remove the banner from the **HTTP header**
- To **remove** the **server banner**
  - Open **$Tomcat/conf** folder
  - Edit **server.xml** file
  - Set server ="" in the **Connector port**

  Example: `<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" Server =" " redirectPort="8443" />`

Header Response with Server Banner

▼ General
Request URL: http:/[redacted]/
Request Method: GET
Status Code: 200 OK
Remote Address: [redacted]
Referrer Policy: no-referrer-when-downgrade

▼ Response Headers view source
Content-Type: text/html;charset=ISO-8859-1
Date: Mon, 22 Jan 2018 08:23:06 GMT
Server: Apache-Coyote/1.1
Transfer-Encoding: chunked

Header Response without Server Banner

▼ General
Request URL: http://[redacted]/
Request Method: GET
Status Code: 200 OK
Remote Address: [redacted]
Referrer Policy: no-referrer-when-downgrade

▼ Response Headers
Content-Type: text/html;charset=ISO-8859-1
Date: Mon, 22 Jan 2018 08:31:44 GMT
Server:
Transfer-Encoding: chunked

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Start Tomcat with Security Manager

- Security manager prevents running **untrusted applets**
- It protects browser from executing **servlets, JSPs, JSP beans, and tag libraries** containing trojan
- To run Tomcat with security manager
  - Add the security manager entries to the JAVA_OPTS environment variable
  - Open catalina.policy file located at <Tomcat_Home>/conf/ in a text editor
  - Alter the SecurityManager permission
  - Start Tomcat with security parameter

Starting Tomcat with Security Manager

$CATALINA_HOME/bin/catalina.sh start -security **(Unix)**

%CATALINA_HOME%\bin\catalina start -security **(Windows)**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

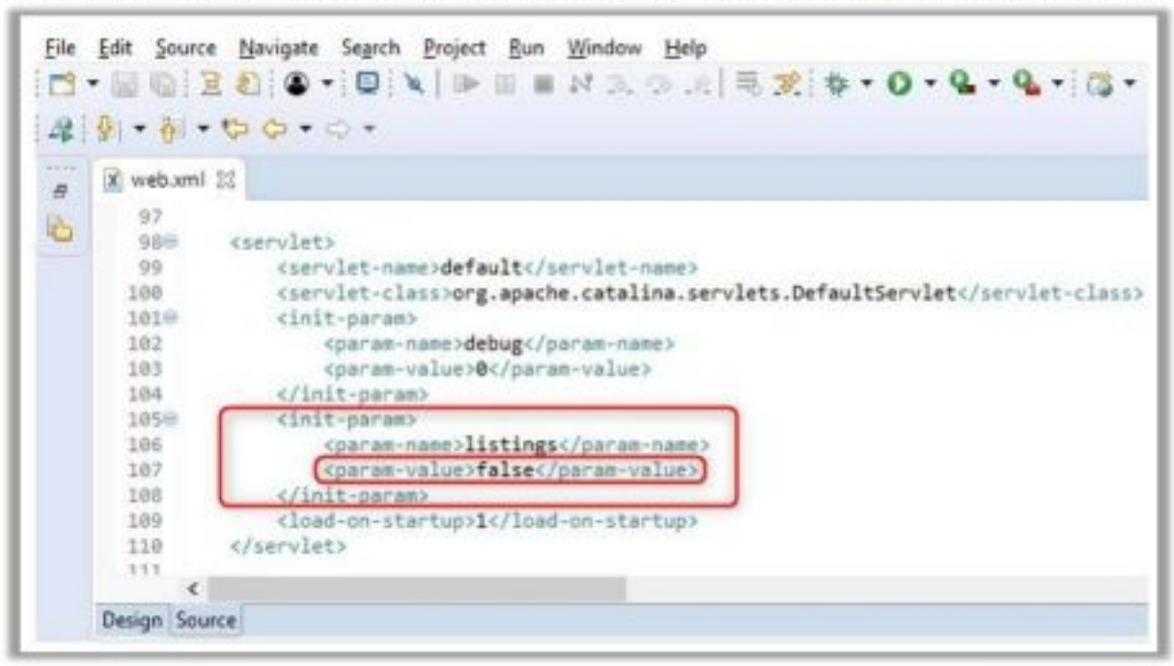
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Configure Default Servlet Not to Serve Index Pages

- Ensure that the default servlet configuration settings do not render index pages in the absence of welcome files
- Edit the **listings** to **false** in **CATALINA_HOME/conf/web.xml** file to configure the server not to serve index pages in absence of welcome files

Default Servlet Configuration to not Serve Index Pages in Absence of Welcome Files

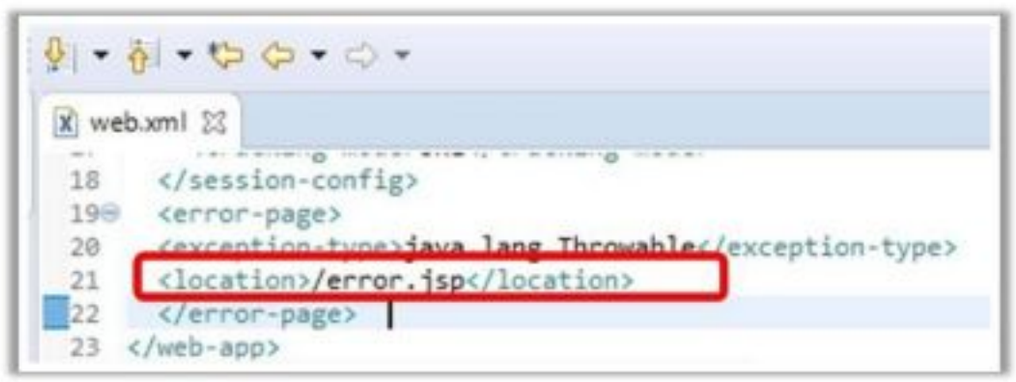


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

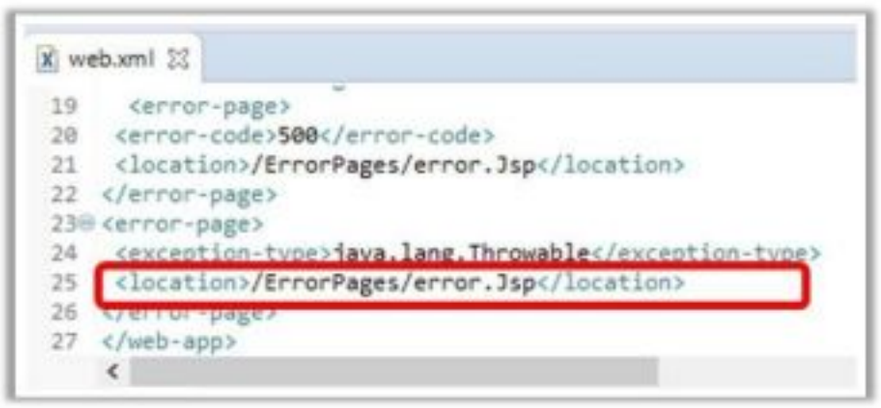
Replace Default Error Page

1. Default error page displays the entire **StackTrace** resulting in sensitive information disclosure
2. Edit **CATALINA_HOME/conf/web.xml** file to replace the default error. This will display a blank page and protects the application from displaying the StackTrace to the user
3. Configure an error page in **CATALINA_HOME/webapps/APP_NAME/WEB-INF/web.xml** to display a custom error page

Configuration to Replace Default Error Page



Configuring Custom Error Page in web.xml



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Replace Default server.xml

- Replace the **server.xml** with **server-minimal.xml** which contains the minimum required configuration

- **To replace server.xml**
  - Rename **server.xml** file present in CATALINA_HOME/conf/ to /originalserver.xml
  - Rename **server-minimal.xml** file present in CATALINA_HOME/conf/ to server.xml

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Protect Shutdown Port

- Tomcat listens to **8005 port** (default) for **shutdown commands**
- When running a publicly accessible server, it should be ensured that external access to the shutdown port is prevented by using a firewall
- To prevent unintended shutting down of Tomcat services, disable the shutdown port
- To disable the shutdown port
  - Edit the port attribute to -1 in the server.xml file
- Setting **port attribute** to **-1** ensures that Tomcat is shutdown only by the user who owns the Tomcat process
- Alter the shutdown command by changing the **shutdownphrase** in the **CATALINA_HOME/conf/server.xml** file
- Ensure that Tomcat users only have read rights for server.xml file

Disabling Shutdown Port



Altering Shutdown Command



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Restrict Access to Tomcat Manager Applications

- Make sure to restrict the IP addresses that can access the Tomcat Manager or administrative web applications
- **To enable connections only from the localhost (127.0.0.1) on the Tomcat server**
  1. Edit context element of **${tomcat_home}/conf/Catalina/localhost/admin.xml**
  2. Edit context element of **${tomcat_home}/conf/Catalina/localhost/manager.xml**

```
web.xml   server.xml   context.xml
25      -->
26  <Context antiResourceLocking="false" privileged="true">
27  <Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="127\.0\.0\.1"/>
28  </Context>
```

- **To enable connections using host name use RemoteHostValve**

```
context.xml
40 <Context>
41 <Valve className="org.apache.catalina.valves.RemoteHostValve" allow=".*\.ucar\.edu" />
42 </Context>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Protecting Resources with Realms

- **Realm** is a database of usernames and passwords for authentication of web applications
- server.xml file has many realms
- Make sure to enable useful realms and others should be disabled
- **LockOutRealm** prevents **brute force attacks** and **denial of service attacks** by locking users, with the number of failed login attempts

Example: server.xml to implement lock out functionality to a UserDatabase Realm

```
server.xml
160 <Realm className="org.apache.catalina.realm.LockOutRealm" >
161 <Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName="UserDatabase"/>
162 </Realm>
Design  Source
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Store Passwords as Digest

- The passwords are stored as clear text in the **${tomcat_home}/conf/tomcat-users.xml** file
- To protect the passwords, store the passwords in **digested form**
  - Generate the **digested form** of the password
  - Update the generated password in **tomcat-users.xml**
  - Include **<Realm>** element under the **<host>** element of the **server.xml** file to implement digested passwords

Password Stored as Clear Text

tomcat-users.xml

```
53  <role rolename="Configuser"/>
54  <role rolename="HRmanager"/>
55  <role rolename="admininistrator"/>
56  <user username="sysadmin" password="pass@$123"
57  roles="manager,admin"/>
58  <user username="cataloguser" password="qwerty@123"
59  roles="Configuser"/>
60  </tomcat-users>
```

Generating Digest Form of the Password

```
> cd ${tomcat_home}/bin
>./digest.sh -a SHA pass@123
> pass@123:aa04ew2sfaae76c2a7dc5e25ex18c505e58f12d7
```

Editing server.xml File

server.xml

```
122   unpackWARs="true" autoDeploy="true" xmlValidation="false"
123 xmlNamespaceAware="false">
124   <Realm className="org.apache.catalina.realm.MemoryRealm" digest="SHA" />
125 </Host>
```

Design | Source

Password Stored in Digest Form

tomcat-users.xml

```
53  <role rolename="Configuser"/>
54  <role rolename="HRmanager"/>
55  <role rolename="admininistrator"/>
56  <user username="sysadmin" password="pass@$123"
57  roles="manager,admin"/>
58  <user username="cataloguser"
59  password="5413ee24723bba2c5a6ba2d0196c78b3ee4628d1"
60  roles="Configuser"/>
61  </tomcat-users>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Do Not Run Tomcat as Root

- Do not run **Tomcat** as root. Create a user with the least required permissions and run Tomcat with the created user account
- Specify the following privileges to the created user
  - Read/Write permission to **${tomcat_home}** directory and subdirectories
  - Read permissions to **data directories**

Running Tomcat as User: Unix / Linux

- Create Tomcat User

```
sudo adduser tomcatUser
```

- Download and unpack Tomcat

```
sudo cp apache-tomcat-8.0.33.tar.gz /opt cd /opt
sudo tar zxvf apache-tomcat-8.0.33.tar.gz
```

- Change ownership to created user

```
sudo chown -R tomcat: tomcatUser /opt/apache-tomcat-8.0.33
```

- Run Tomcat as a created user

```
sudo -u tomcat /opt/apache-tomcat-8.0.33/bin/catalina.sh run
```

Running Tomcat as User: Windows

- Create unprivileged user account using **Administrative Tools →Local Security Policy → Local Policies → User Rights Assignment → Log on as a service**
- Ensure to set **Apache Tomcat** service to run as the created user

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Configure Restricted Datasets

1. While restricting access to datasets, create separate users with **restrictedDatasetUser** roles
2. Users with a **restrictedDatasetUser** role can use **access non-HTTP URL**, which can give rise to **session hijacking**
3. Users with a **restrictedDatasetUser** role should not be assigned roles with privileges to access the security features of Tomcat

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Session Handling using App Mode in Tomcat

Session Timeout

- Session timeout for every web application must be set to **minimum required duration**
- To set the timeout edit the **CATALINA_HOME/conf/web.xml** file with following code

web.xml

```
585    <session-config>
586        <session-timeout>20</session-timeout>
587    </session-config>
```

HttpOnly Flag

- This flag can be activated with the following configuration option that can be set in the CATALINA_HOME/conf/context.xml
- If HttpOnly flag is set to true then it may break the application functionality, if access to the session cookie via javascript is necessary

Syntax:

context.xml

```
37
38 <Context useHttpOnly="true">
39
```

CSRF Prevention Filter

- It helps prevent **Cross Site Request Forgery**
- CSRF filter generates a nonce and stores it in a session. It encodes the URLs with the generated nonce. On receiving another request for the URL the received nonce is matched with the sessions nonce. If the two do not match the request is considered forged.
- CSRF Filter can be configured in file CATALINA_BASE/conf/web.xml at Globally . The global setting can overwrite by writing in WEB-INF/web.xml

web.xml | context.xml

```
548
549    <filter-mapping>
550        <filter-name>httpHeaderSecurity</filter-name>
551        <url-pattern>/*</url-pattern>
552        <dispatcher>REQUEST</dispatcher>
553    </filter-mapping>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Session Handling using App Mode in Tomcat (Cont'd)

Disable Session URL rewriting

- This flag is used to disable the session id displayed in the URL with the following configuration
- CATALINA_HOME/conf/ web.xml with following code:

```
585    <session-config>
586        <session-timeout>20</session-timeout>
587        <tracking-mode>COOKIE</tracking-mode>
588    </session-config>
```

Secure Cookie Flag

- This flag will force the transmission of a cookie only by SSL, with the following configuration
- CATALINA_HOME/conf/ web.xml with following code

```
585    <session-config>
586      <cookie-config>
587    <secure>true</secure>
588      </cookie-config>
589    </session-config>
```

Replace the Default Session Name

- The default session name is JSESSIONID. The following configuration could be used to change the default name
- **CATALINA_HOME/conf/ web.xml** with following code:

```
585    <session-config>
586      <cookie-config>
587      <name>MySESSIONID</name>
588      </cookie-config>
589    </session-config>
```

Change the Session Id length

- The default session id length is 128bit / 16bytes. The-following configuration could be used to change the default value
- **CATALINA_HOME/conf/context.xml** with following code

```
19
20  <Context >
21    <Manager pathname="" sessionLength="32" />
22
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Role Based Security

- Tomcat supports **role based authorization**

- To implement role based authorization
  1. Create users and roles in /conf/tomcat-user.xml
  2. Define Security role in web.xml

Example : Setting Roles and Users in Tomcat-user.xml

```
67    <role rolename="tomcatUser"/>
68    <role rolename="roleA"/>
69    <role rolename="userMain"/>
      <user username="tomcatUser" password="tomcatusrpwd" roles="tomcatUser"/>
      <user username="roleA" password="tomcatusrpwd" roles="roleA"/>
      <user username="Mixed" password="tomcatusrpwd" roles="tomcatUser,roleA"/>
      <user username="Tom" password="tompwd" roles="userMain"/>
74  </tomcat-users>
```

Example : Defining Security Role in web.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2    <web-app>
3    <welcome-file-list>
4    <welcome-file>index.html</welcome-file>
5    </welcome-file-list>
6    <security-role>
7  <description>The role use Manager Application </description>
8  <role-name>userMain</role-name>
9  </security-role>
10 <security-constraint>
11 <web-resource-collection>
   -pattern> /protected/* </url-pattern>
   b-resource-collection>
   h-constraint>
   e-name>userMain </role-name>
   th-constraint>
   curity-constraint>
   b-app>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Securing Tomcat at Network Level

Prevent The Connectors:

- Prevent the application from listening on all interfaces/IP addresses available on the server system. Instead, the IP address must be specified
- Edit **CATALINA_HOME/conf/server.xml** file to restrict the listening interfaces
- Review every connecter and specify the correct IP addresses

Syntax:

```
<connector PORT="TCP_PORT"
address="Listen_IP_Address"…….
```

Tomcat Port Connection:

- Tomcat has many ports for network connection. To establish a connection a port works as an identifier. These ports need to be configured correctly
- For TCP default ports are **8080** and **8443**. These ports are very important and need to be configured correctly
- By opening **CATALINA_HOME/conf/server.xml** file we can see every connector configuration for the correct/desired port assignment. Remove unnecessary or unused ports or connections

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Securing Tomcat at Network Level (Cont'd)

Trustworthy connection using encryption of network connection:

- **SSL configuration:** To secure an application, SSL configuration must be applied. SSL provides **encryption** and **decryption** for communication between two parties over the network. For this, the public key encryption method is used. The following steps are used for Tomcat configuration to use SSL
- **Two different implementations of SSL**:
  - **JSSE Implementation** →If Tomcat native library is installed
  - APR Implementation uses **OpenSSL** engine

- Edit the file **CATALINA_HOME/conf/server.xml** with following code

server.xml

```
88
89  <Connector protocol="org.apache.coyote.http11.Http11Protocol"
90  port="8087" scheme="https" secure="true" SSLEnabled="true"
91  sslProtocol="TLS" keystoreFile="keystore file path"
92  keystorePass="keystore password"/>
93
```

Design | Source

- Add the following in **CATALINA_HOME/webapps/manager/WEB_INF/web.xml** under <security-constraint> tags

tomcat-users.xml | web.xml

```
39    <display-name></display-name>
40    <user-data-constraint><description></description>
41    <transport-guarantee>CONFIDENTIAL</transport-guarantee
42    </user-data-constraint>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Securing Tomcat at Network Level (Cont'd)

- **Use valve to filter by IP/hostname that allows only a subset of machines to connect**
  - Any one of the following codes can be added in the Context tag in **CATALINA_HOME/conf/Catalina/localhost/context.xml**

Example 1

```
<!-- allow only LAN IPs to connect to the manager webapp -->
<!-- contrary to the current Tomcat 7.0 documentation the value for allow is not
a regular expression -->
<!-- future versions may have to be specified as 192\.168\.1\.* -->
```

context.xml

```
41
42          <Valve className="org.apache.catalina.valves.RemoteAddrValve"allow="192.168.1.*" /
43 </Context>
```

Design | Source

Example 2

```
connect to the manager webapp -->
<!-- contrary to the current Tomcat 7.0 documation the value for allow is not a
regular expression -->
<!-- future versions may have to be specified as *\.localdomain\.com -->
```

context.xml

```
41
42          <Valve className="org.apache.catalina.valves.RemoteAddrValve"
43          allow="localdomain.com" />
```

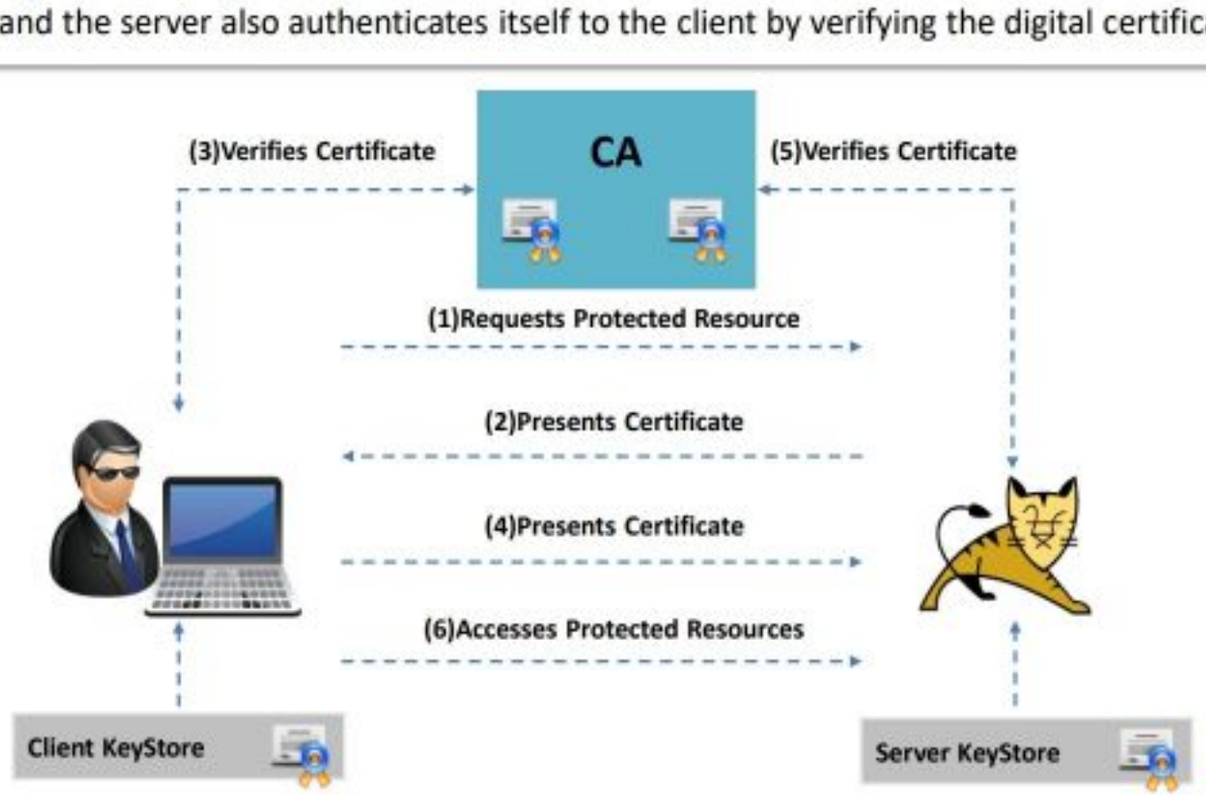
Design | Source

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Securing Tomcat at Network Level (Cont'd)

Mutual Authentication

- **Certificate-based mutual authentication** provides the mechanism to authenticate two parties by providing and verifying the certificate to each other. Web browsers authenticate themselves to a web server and the server also authenticates itself to the client by verifying the digital certificate



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Securing Tomcat at Network Level (Cont'd)

- **Create Client Side KeyStore**

  command: `$JAVA_HOME/bin/keytool -genkey -alias client -keyalg RSA -keystore client.jks`
- **Export Client Side Certificate File**

  command: `$JAVA_HOME/bin/keytool -export -keystore client.jks -alias client -file client.cer`
- **Import client.cer to server trustStore**

  command: `$JAVA_HOME/bin/keytool -importcert -file client.cer -keystore truststore.jks -alias client`
- **Convert the client.jks to PKCS12 format**

  command:
  ```
  $JAVA_HOME/bin/keytool -importkeystore -scrkeystore client.jks - destkeystore client.pfx -
  srcstoretype JKS -deststoretype pkcs12 - deststorepass "XXX" -alias client -destalias client
  ```
- **Import the client.pfx file to client side browser**
- **Configure server.xml**
- **Edit the file CATALINA_HOME/conf/server.xml with following code**

server.xml

```
89  <Connector protocol="org.apache.coyote.http11.Http11Protocol"
90  port="443" scheme="https" secure="true" SSLEnabled="true"
91  sslProtocol="TLSv1.2"
92  sslEnabledProtocols="TLSv1.1, TLSv1.2" clientAuth="true"
93   keystoreFile="keystore file path" keystorePass="keystore password"
94  truststoreFile="truststore file path" truststorePass="truststore password"
```

Design | Source

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Java Runtime Security Configurations

- **Java Security Manager**
  - Java security manager can be configured in file **CATALINA_HOME/conf/Catalina.policy**
  - Once the file is configured Tomcat will start with **SecurityManager** in place of security option
- **Java Package Access**
  - Tomcat has a feature to restrict package access
  - A user trying to access a package without access permission gets access exception
  - To modify **package.access** alter the **package.access** list in **CATALINA_BASE/conf/Catalina.properties**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Tomcat General Security Setting

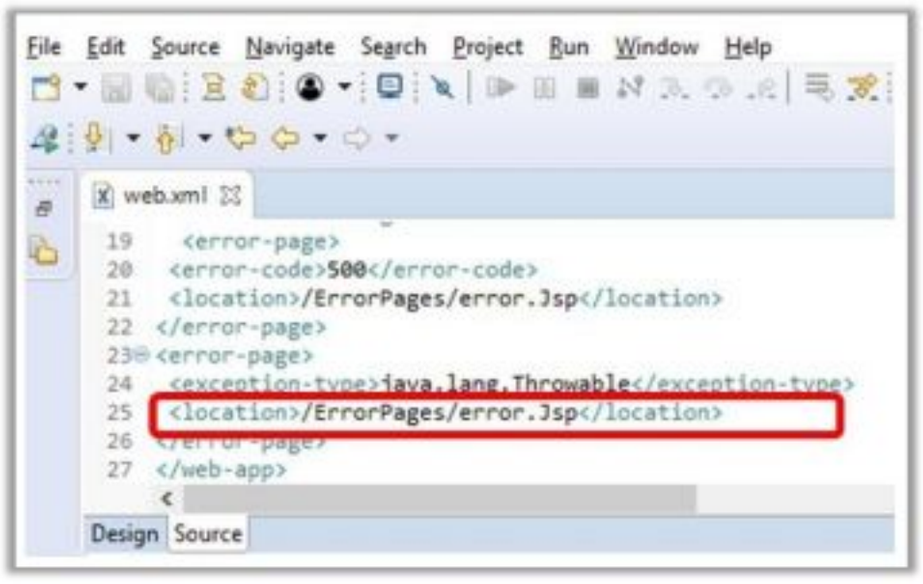
- Do not make unnecessary changes in the default settings of Tomcat configurations

- Remove **unwanted, default, servlet examples**, etc. from Tomcat **webapp directory**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Tomcat General Security Settings (Cont'd)

- **Custom Error Page**
  - It is the best practice to include a custom error page in the web application. The following code must be added in **CATALINA_HOME/webapps/webapp/WEB-INF/web.xml**



- **Automatic Deployment**
  - Automatic deployment must be **disabled** to prevent malicious applications from deploying
  - This can be done by editing the file **CATALINA_HOME/conf/server.xml**
  - Change the following line in server.xml file

```
<.. ……..autoDeploy="false">

<.. ……..DeployOnStartup="false">
```

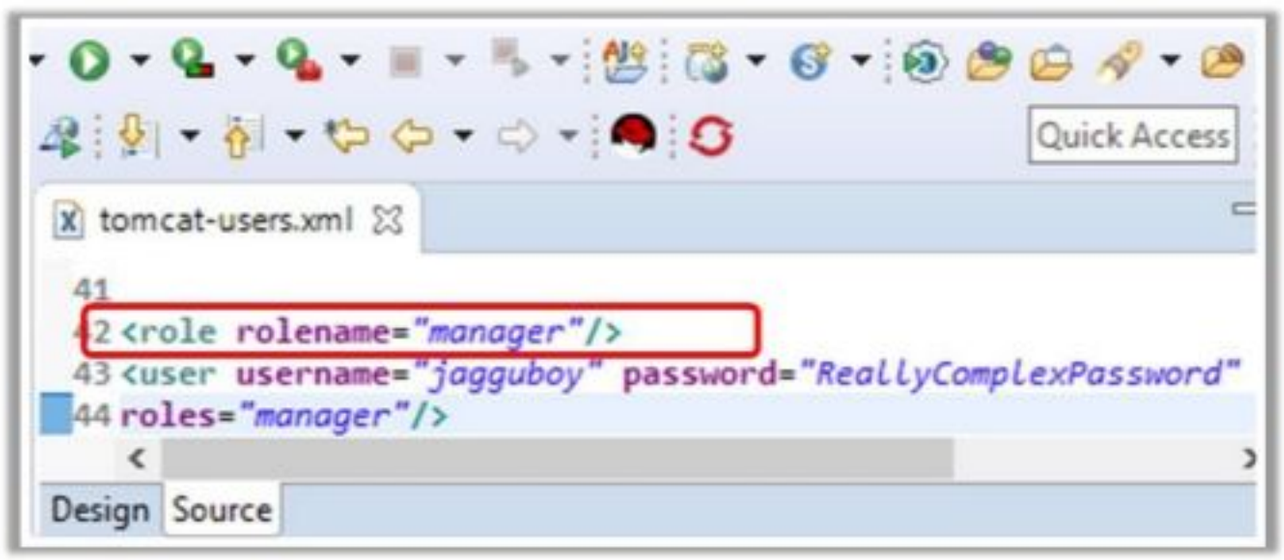
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Tomcat General Security Settings (Cont'd)

- **Create Webapp**
  - A manager Webapp can be created by adding a new role and user into the **CATALINA_HOME/conf/tomcat-users.xml** file as follows:



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Tomcat General Security Settings (Cont'd)

- **Rename the Manager Webapp as follows**
  - In the example below the new name is "**sample**"

```
1: Move CATALINA_HOME/conf/Catalina/localhost/manager.xml to
2: CATALINA_HOME/conf/Catalina/localhost/sample.xml
```

```
1: Update the docBase attribute within CATALINA_HOME/conf/Catalina/localhost/sample.xml to
2: {catalina.home}/server/webapps/foobar
```

```
1: Move CATALINA_HOME/server/webapps/manager to CATALINA_HOME/server/webapps/sample
```

- **Run Tomcat with a Security Manager that controls the access to server resources**
  - It is a good practice to start Tomcat with "`-security`" parameter

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Verify Trace Element Setting in server.xml

- The tracing process is accountable for all activities accomplished by a web page on a web server and is enabled mainly to troubleshoot issues during the development of an application
- However, leaving trace enabled after deployment of the application may expose critical information and allow attackers to alter and manipulate the application
- Set **<Connector URIEncoding="UTF ..... allowTrace="false" />.** Disabling trace element prevent the attackers from gaining information from the trace

Vulnerable Configuration

```
server.xml
97  <Connector URIEncoding="UTF-8"
98  acceptCount="100"
99  sslProtocol="TLS"
100 allowTrace="true">
101 </Connector>
Design Source
```

Secure Configuration

```
server.xml
97  <Connector URIEncoding="UTF-8"
98  acceptCount="100"
99  sslProtocol="TLS"
100 allowTrace="false">
101 </Connector>
Design Source
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Verify CustomError Settings in web.xml

- Attackers can use this information to make an educated guess and craft attack techniques to compromise the security of the application
- The attackers may use the exception details to **perform attacks** or **manipulation of information** in the application
- The user will be redirected to any generic web page in the event of any error

The following code from web.xml file shows a secure way of setting custom error:

Secure Configuration

```
*web.xml
18
19
20  <error-page>
21    <error-code>404</error-code>
22    <location>**/error.html**</location>
23  </error-page>
24  <error-page>   <error-code>404</error-code>
25    <location>**/error.html**</location>
26  </error-page>
27  <error-page>
28    <exception-type>java.lang.Throwable</exception-type>
29    <location>**/error.html**</location>
30  </error-page>
Design Source
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Verify maxPostSize Setting

1. The **maxPostSize** attribute of httpRuntime element, restricts attackers from uploading files with large sizes to the server
2. Allowing users to upload files with large sizes may put the application under risk of **denial-of-service** attacks
3. Set **<connector… maxPostSize="6291456" />** the limit can be disabled by setting this attribute to less than zero, like **-1**

Vulnerable Configuration

server.xml

```
103
104 <Connector port="8080" protocol="HTTP/1.1"
105 connectionTimeout="20000"
106 redirectPort="8443"
107 maxPostSize="0" />
108
```

Secure Configuration

server.xml

```
102
103
104 <Connector port="8080" protocol="HTTP/1.1"
105 connectionTimeout="20000"
106 redirectPort="8443"
107 maxPostSize="6291456" />
```

Design | Source

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Tomcat Security Checklist

- The following default settings must not be changed in server.xml
  - **allowTrace="false"**
  - **Privileges="false"**
  - **crossContext="false"**
  - **allowLinking="false"**
- Make sure to change **DefaultServlet configuration** with **read only set to false**
- Make sure the following are set to false in Startup script
  - Dorg.apache.catalina.connector.RECYCLE_FACADES=false
  - Dorg.apache.catalina.connector. CoyoteAdapter.ALLOW_BACKSLASH=false
  - Dorg.apache.tomcat.util.buf. UDecoder.ALLOW_ENCODED_SLASH=false
  - Dorg.apache.coyote.USE_CUSTOM_STATUS_MSG_IN_HEADER=false

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

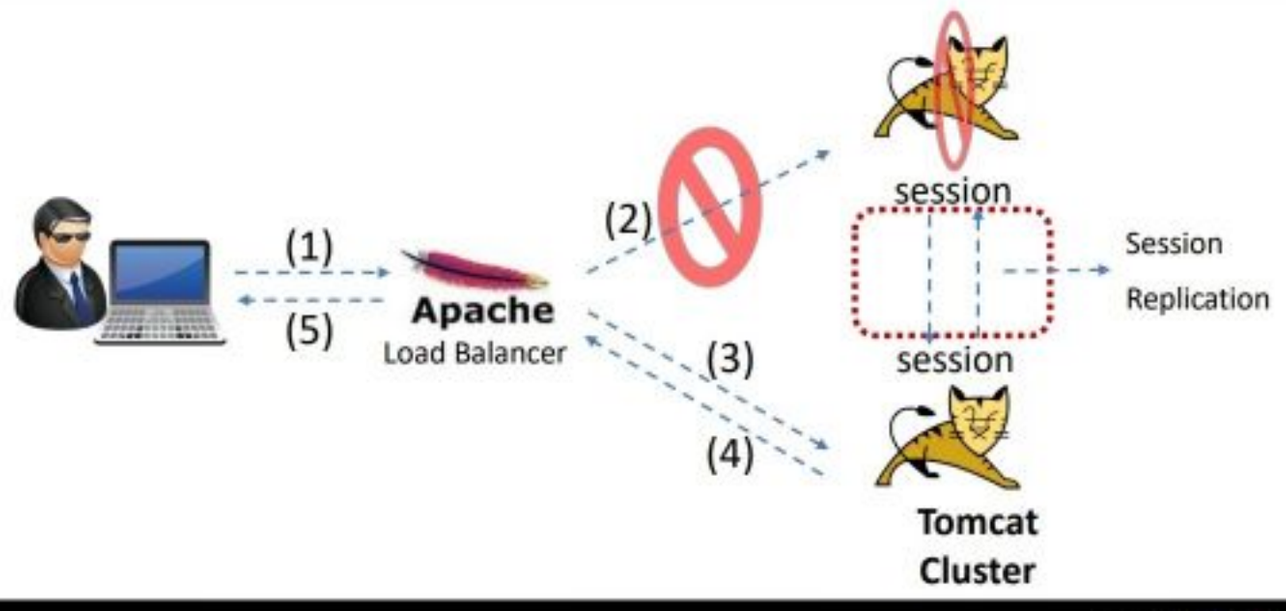
Checklist for Security Configuration in server.xml File in Apache Tomcat

- **Port Disabling**
  - Avoid malicious attacks by setting Port on the server element to **(-1)** disable
- **Security Lifecycle Listener**
  - Linux :By using **Security Lifecycle Listener** we can prevent Tomcat from running as root user
- **Specifying Interfaces for connectors**
  - We can specify address attribute of the connector tag or element thereby decrease the ways of attacks
- **Disable allowTrace Feature**
  - The **AllowTrace** feature enables the debugging and tracing feature which becomes a source of sensitive information discloser
- **Disable sslEnableProtocols**
  - By exploiting this (**sslEnableProtocols**) vulnerability a user can gain access to sensitive information like password, cookies etc.
  - It is recommended that **sslEnableProtocols** is **disabled**
- **Disabling Directory Listing**
  - When the directory listing feature is enabled the application displays the entire directory structure of the web application
  - Displaying directories containing a large no of files results in denial-of-service attack. To disable directory listing set **DefaultServlet listings** to **false**
- **Disable Automated Deployment(if not in use)**
  - Ensure that the host attributes (autoDeploy, deployOnStartup, and deployXML) are disabled if not in use
- **Avoid running Tomcat under root user**
  - Do not run Tomcat under a user account with administrative privileges
  - Ensure to create a user account with minimum required OS permissions for running the Tomcat process

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Tomcat High Availability

- The system requires to be in **zero downtime** effectively while failure or disaster occurs
- An alternative system is the way to achieve immediate availability
- With **Session Replication**, clients don't have to reauthenticate while one of the server instance is down



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Tomcat High Availability (Cont'd)

- **Tomcat Clustering Configuration:**
  - Edit the file **CATALINA_HOME/conf/server.xml,** add the following code in the <**engine**> section

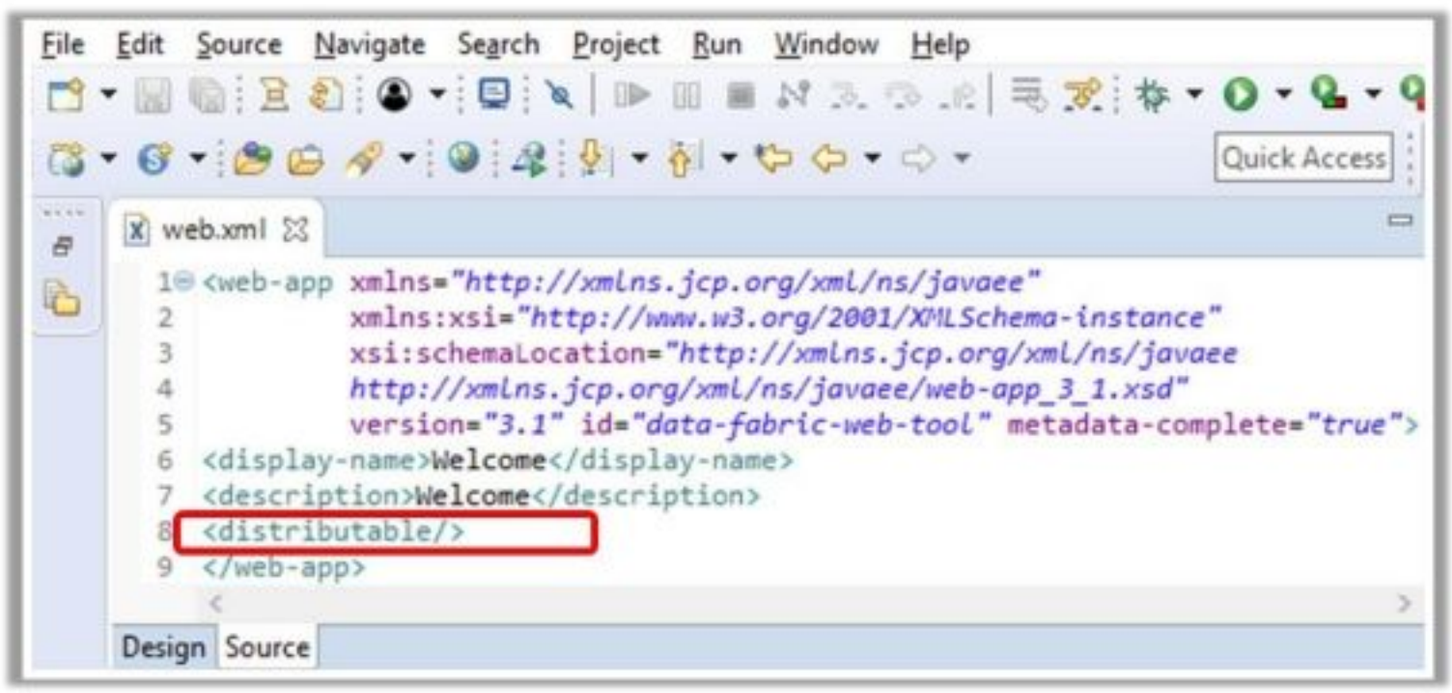
server.xml

```
116     -->
117     <Engine defaultHost="localhost" name="Catalina">
118     <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster" channelSendO
119     <Manager className="org.apache.catalina.ha.session.DeltaManager" expireSessio
120      notifyListenersOnReplication="true"/>
121     <Channel className="org.apache.catalina.tribes.group.GroupChannel">
122     <Membership className="org.apache.catalina.tribes.membership.McastService"
123     address="228.0.0.4"    port="45564"
124               frequency="500"  dropTime="3000"/>
125               <Receiver className="org.apache.catalina.tribes.transport.nio.Nio
126               address="auto" port="4000" autoBind="100"
127               selectorTimeout="5000" maxThreads="6"/>
128               <Sender className="org.apache.catalina.tribes.transport.Replicati
129               <Transport className="org.apache.catalina.tribes.transport.nio.Po
130               </Sender>
131               <Interceptor className="org.apache.catalina.tribes.group.intercep
132               <Interceptor className="org.apache.catalina.tribes.group.intercep
133               </Channel>
134               <Valve className="org.apache.catalina.ha.tcp.ReplicationValve" fi
135               <Valve className="org.apache.catalina.ha.session.JvmRouteBinderVa
136               <Deployer className="org.apache.catalina.ha.deploy.FarmWarDeploye
137               tempDir="/tmp/war-temp/"deployDir="/tmp/war-deploy/"
138               watchDir="/tmp/war-listen/"
139               watchEnabled="false"/>
140               <ClusterListener className="org.apache.catalina.ha.session.Cluste
141               </Cluster>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Tomcat High Availability (Cont'd)

- **Tomcat Clustering Configuration:**
  - Edit the file **APPLICATION/WEB_INF/web.xml** , add the code **<distributable />**



- **Restart Tomcat Services**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

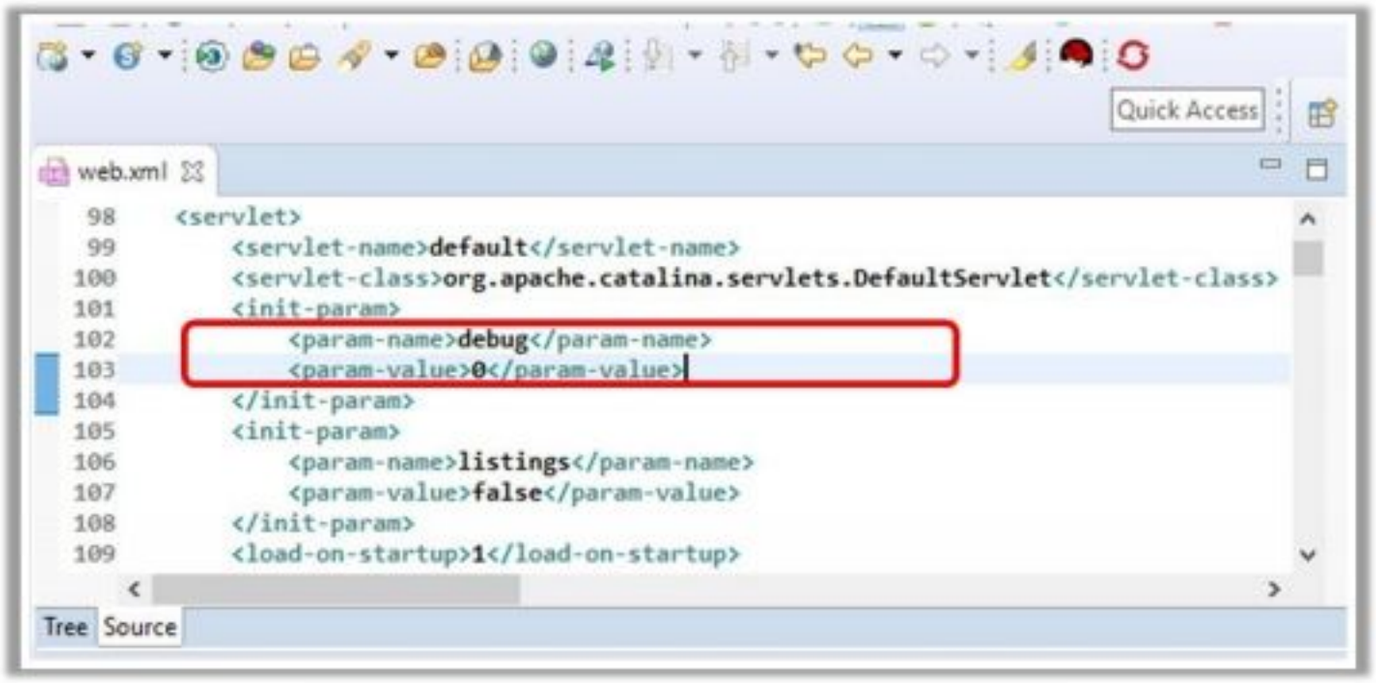
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Best Practices for Securing Tomcat

- Delete everything from the path **CATALINA_HOME/webapps**
- Delete everything from **CATALINA_HOME/server/webapps**
- Delete **CATALINA_HOME/conf/Catalina/localhost/host-manager.xml**
- Ensure that the default web servlet configuration does not serve the index pages in the absence of a welcome page

The following code should be implemented in CATALINA/conf/web.xml:



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Best Practices for Securing Tomcat (Cont'd)

- Replace the version string of HTTP error messages with **CATALINA_HOME/server/lib/catalina.jar** and an updated **ServerInfo.properties** file
- Replace the default error page code in **CATALINA_HOME/conf/web.xml** with the following code



- Rename **CATALINA_HOME/conf/server.xml** to **CATALINA_HOME/conf/server-original.xml**
- Rename **CATALINA_HOME/conf/server-minimal.xml** to **CATALINA_HOME/conf/server.xml**
- Replace the server version string from HTTP headers in server responses, by adding the server keyword in your Connectors in **CATALINA_HOME/conf/server.xml**



- Clear all text passwords from **CATALINA_HOME/conf/server.xml**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Ensuring Security in Oracle

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Oracle Database General Security Overview

- Managing oracle database installation and configuration securely
- Managing user accounts securely for your site
- Control access to data
- Securing data traversal over the network

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Methods of Authentication in Oracle

- **To verify users' identity Oracle provides these authentication methods. Users can use a combination of these methods:**

Operating System	Authentication by **Operating System**
Network and LDAP Directories	Authentication by **Network**
Database	Authentication by **Oracle Database**
Multitier System	**Multitier** Authentication and Authorization
Secure Socket Layer Usage	Authentication of **Database Administrators**
Database Administrators	Authentication of **Database Administrators**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

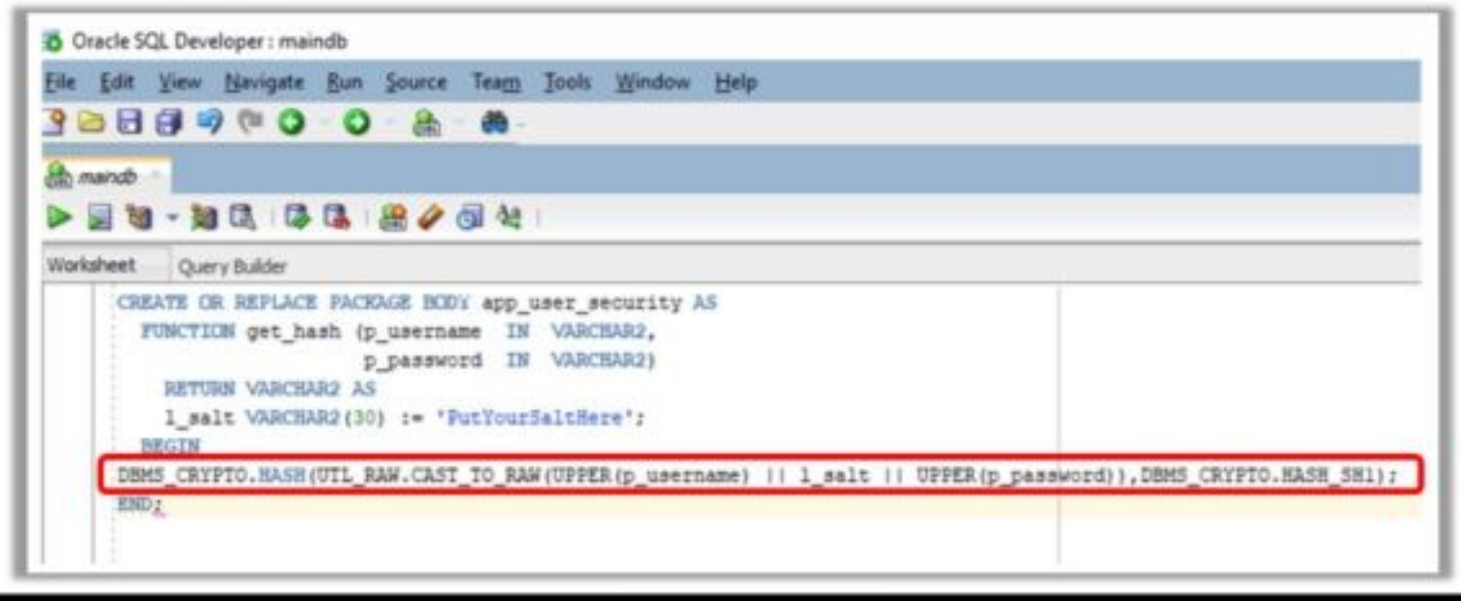
Authentication by Oracle Database

- To implement Oracle database authentication, DBA must create users and associate them with a password to establish a connection
- Set the **SQLNET.ALLOWED_LOGIN_VERSION** parameter in the server **sqlnet.ora** file to identify the authentication protocols allowed by the client or database

Oracle Database Authentication Supports:

Password Encryption

- **DBMS_CRYPTO** Package is used for encryption and hashing

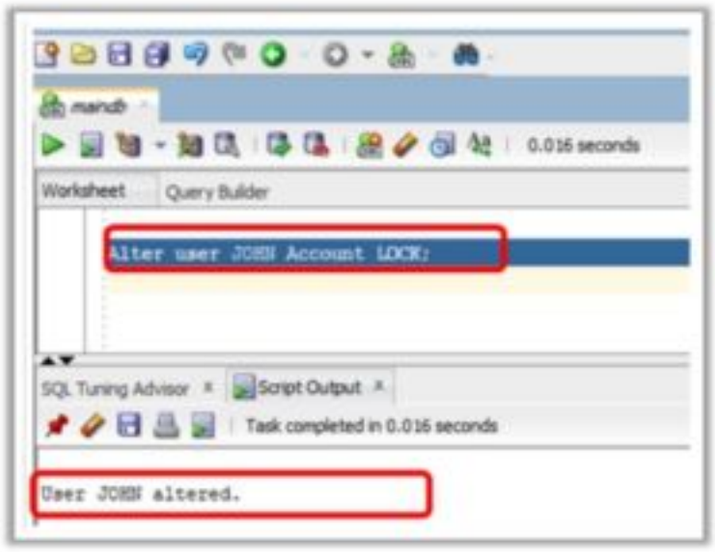


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Authentication by Oracle Database (Cont'd)

- **Account Locking**
  - DBA can Lock accounts manually for a particular user command to lock a specific user



- **Password Lifetime and Expiration**
- **Password History**
- **Password Complexity Verification**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Oracle Security Features

Case Sensitive Passwords

- While creating Database via DBCA, a user is prompted to upgrade to **"New Security Standards"**. If chosen to do so, the created passwords will be case sensitive
- When you try to login, considering passwords are not case sensitive, an error will be prompted
- **For example,**

  SQL> conn scott/TIGER

  ERROR:

  ORA-01017: invalid username/password; logon denied

  Warning: You are no longer connected to ORACLE
- To temporarily disable the case sensitive

  set **sec_case_sensitive_logon = false**

  Example: SQL> alter system set sec_case_sensitive_logon = false;

  System altered

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

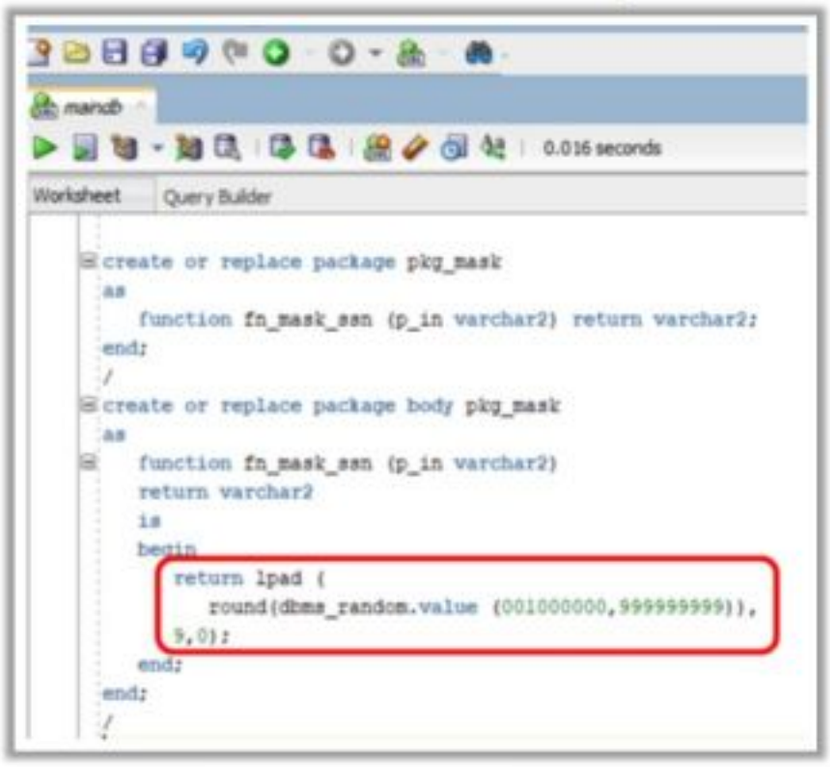
@ErrorCatchers

Oracle Security Features (Cont'd)

Data Masking

- Converting sensitive data to some meaningless text in the non-production database, is a process known as data masking
- **Remap Function:** At the time of import process data will change

Following is a Function which is used for Remap Data



```
create or replace package pkg_mask
as
   function fn_mask_ssn (p_in varchar2) return varchar2;
end;
/
create or replace package body pkg_mask
as
   function fn_mask_ssn (p_in varchar2)
   return varchar2
   is
   begin
      return lpad (
         round(dbms_random.value (001000000,999999999)),
      9,0);
   end;
end;
/
```

```
    ACCNO ACCNAME                          ACCSSN_No
 --------- ------------------------------
        1 Kelvin                          123456789
        2 Robert                          234567890

Here is table Accounts user wants to mask coloumn
ACC_SSN. A method Data Dump.At the time of
exporting have to use
Implement remap_data  parameter to mask the data.

$ expdp scott/tiger tables=scott.accountmaster
dumpfile=
   accounts.dmp
directory=tmp_dirremap_data=accounts.acc_ssn:pkg_m
ask.fn_mask_ssn
/
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Default Database Installation and Configuration Security

- Oracle database contains **built-in protections** for managing different sections of DB. Some settings are while installing the database. Following some of the settings while installation will help protect the database and database dependent applications

Default Security Setting for New Database Creation

- Default **Auditing** setting enables investigating suspicious activity
- Creating **Strong Passwords**, using a combination of upper case, lower case and special characters and numbers in the password, password length should be between 12 and 30 characters
- Remove the **CREATE EXTERNAL JOB privilege** from the **PUBLIC** role
- While installation set **Security-related Initialization** and **Profile Parameter** settings

Default Security Settings for Initialization and Profile Parameters

Setting	10g Default	11g Default
AUDIT_TRAIL	NONE	DB
07_DICTIONARY_ACCESSIBILITY	FALSE	FALSE
PASSWORD_GRACE_TIME	UNLIMITED	7
PASSWORD_LOCK_TIME	UNLIMITED	1
FAILED_LOGIN_ATTEMPTS	10	10
PASSWORD_LIFE_TIME	UNLIMITED	180
PASSWORD_REUSE_MAX	UNLIMITED	UNLIMITED
PASSWORD_REUSE_TIME	UNLIMITED	UNLIMITED
REMOTE_OS_ROLES	FALSE	FALSE

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Managing User Accounts Securely for the Site

Securing Oracle Database User Account

- On installation, Oracle provides some predefined accounts. The administrator should secure these accounts by the following methods
  - Safeguarding predefined **Database User Accounts**, setting secure password protection
  - Managing **Database Accounts** such as expired and or database account locks for users
  - Password management

- Oracle Database creates a set of predefined accounts at the time of the installation process. The following are some categories
  - Administrative accounts
  - Non-administrative accounts
  - Sample schema user accounts

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Securing User Accounts

Lock and Expire the Default Created Accounts

- Make sure to lock and expire all pre-defined user accounts created while installation. The database configuration assistant by default locks and expires the created accounts
- Use **DBA_*, DBA_ROLES, DBA_SYS_PRIVS, DBA_ROLE_PRIVS, DBA_TAB_PRIVS, DBA_AUDIT_TRAIL** (if standard auditing is enabled), **DBA_FGA_AUDIT_TRAIL** (if fine-grained auditing is enabled) data dictionary views to get user database access details

- Revoke SYS.USER_HISTORY$ table access from all users except SYS and DBA accounts
- Revoke RESOURCE and CONNECT roles from typical application accounts
- Restrict proxy account privileges to CREATE SESSION only
- Maintain roles specific to each database installation
- Ensure that the default user passwords are modified
- Build a complex, long password which is easy to remember
- Make sure to enforce password policy
- Encrypt the column storing the user passwords
- Revoke Execute from **PUBLIC** for **UTL_SMTP, UTL_TCP, UTL_HTTP, UTL_FILE, DBMS_OBFUSCATION_TOOLKIT** packages

Example : Revoke on utl_file

```
SQL> REVOKE execute ON utl_file FROM PUBLIC;
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Securing User Accounts (Cont'd)

- **Implement least privilege principle:** Grant minimum required privileges needed to execute the jobs
  - Restrict SYSTEM and OBJECT privileges granted to database users
  - Restrict users having SYS-privileged connections to the database
  - Restrict users having privileges, like DROP ANY TABLE
  - Restrict users having privileges to create, modify, or drop database objects

  Example : SQL Query to get all users with the DBA role

```
SELECT grantee FROM dba_role_privs WHERE granted_role = 'DBA';
```

- **Restrict giving CREATE ANY EDITION and DROP ANY EDITION privileges:** CREATE ANY EDITION and DROP ANY EDITION privileges should only be assigned to users who perform upgrades

- **Restrict security related privileges:** Privileges like CREATE ANY JOB, BECOME USER, EXP_FULL_DATABASE, and IMP_FULL_DATABASE should be restricted to a few users

- **Restrict non-administrative users from accessing SYS schema owners objects:** To protect data integrity only administrative users should be allowed to alter table rows or schema objects present in SYS schema, because doing so can compromise data integrity

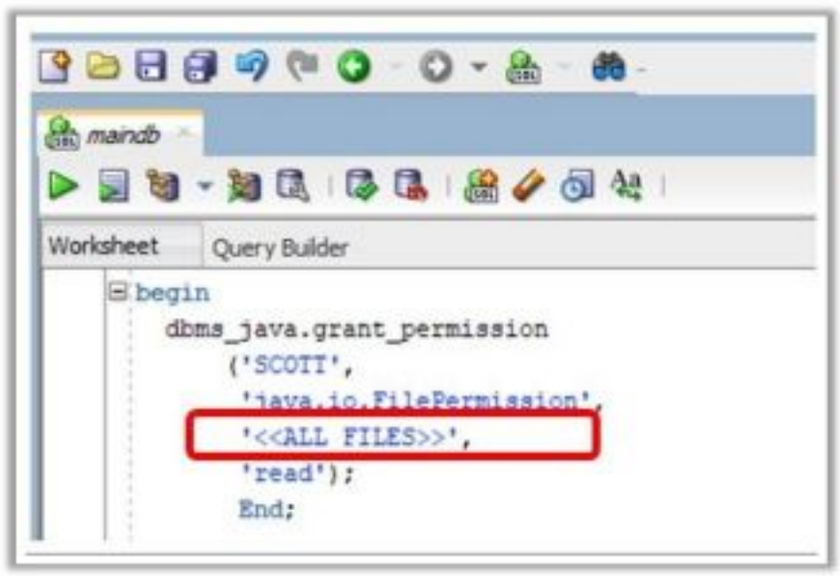
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Securing User Accounts (Cont'd)

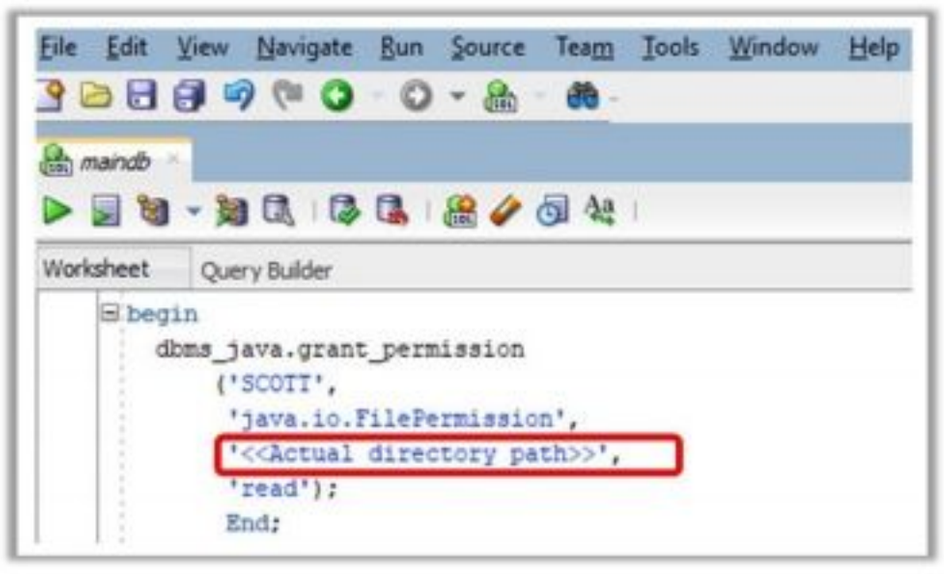
- Restrict giving **EXECUTE** privileges on the DBMS_RANDOM PL/SQL package

- Give limited **users permissions** on run-time facilities

Vulnerable Runtime Call



Secure Runtime Call-Specifying a Directory Path



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Password Management

Password Locking

Parameter	Description
FAILED_LOGIN_ATTEMPTS	Number of failed login attempts before a user account is locked
PASSWORD_LOCK_TIME	Duration in days for locking the account once the failed login attempts exceed

Password Expiration and Aging

Parameter	Description
PASSWORD_LIFE_TIME	No. of days a password is valid before expiring
PASSWORD_GRACE_TIME	No of days after first successful login after the password has expired

Password History

Parameter	Description
PASSWORD_REUSE_TIME	Duration in days for password reuse
PASSWORD_REUSE_MAX	Number of password changes before re-using the current password

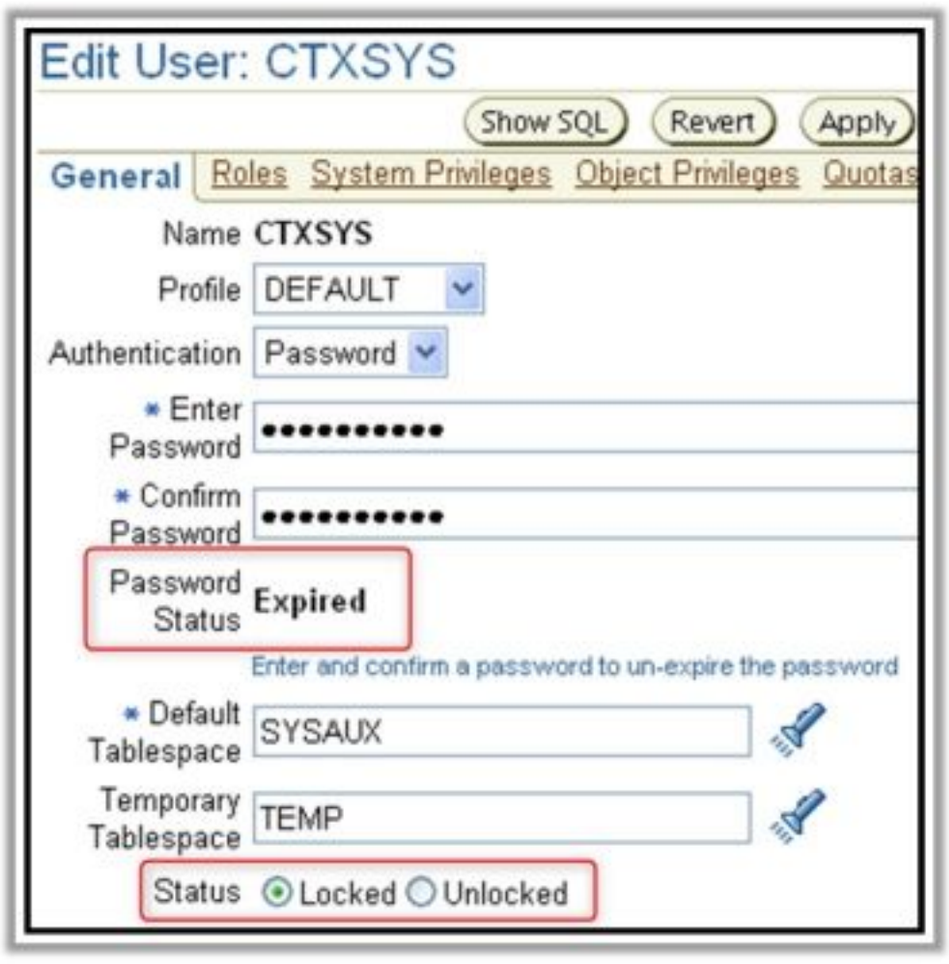
Password Verification

Parameter	Description
PASSWORD_VERIFY_ FUNCTION	PL/SQL function for checking the complexity of password before assigning the password (Should be owned by `SYS User`)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Lock all Expired Accounts

- **DBCA** by default **expires and locks** all accounts other than the following Accounts
  - SYS
  - SYSTEM
  - SYSMAN
  - DBSNMP
- Ensure that you lock and expire all Manually created accounts which are not in use



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Assign Users to Password Profile

- **Assign a password profile** to a user by editing the user



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Disable Remote Operating System Authentication

- Since while implementing **Remote Authentication** the database user is authenticated externally on remote system, make sure to disable remote authentication unless you trust all clients using remote authentication

- To disable remote **Operating System Authentication**, ensure that the **REMOTE_OS_AUTHENT** instance initialization parameter is **FALSE**

- Remote Operating System Authentication is disabled by default

```
REMOTE_OS_AUTHENT = FALSE
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

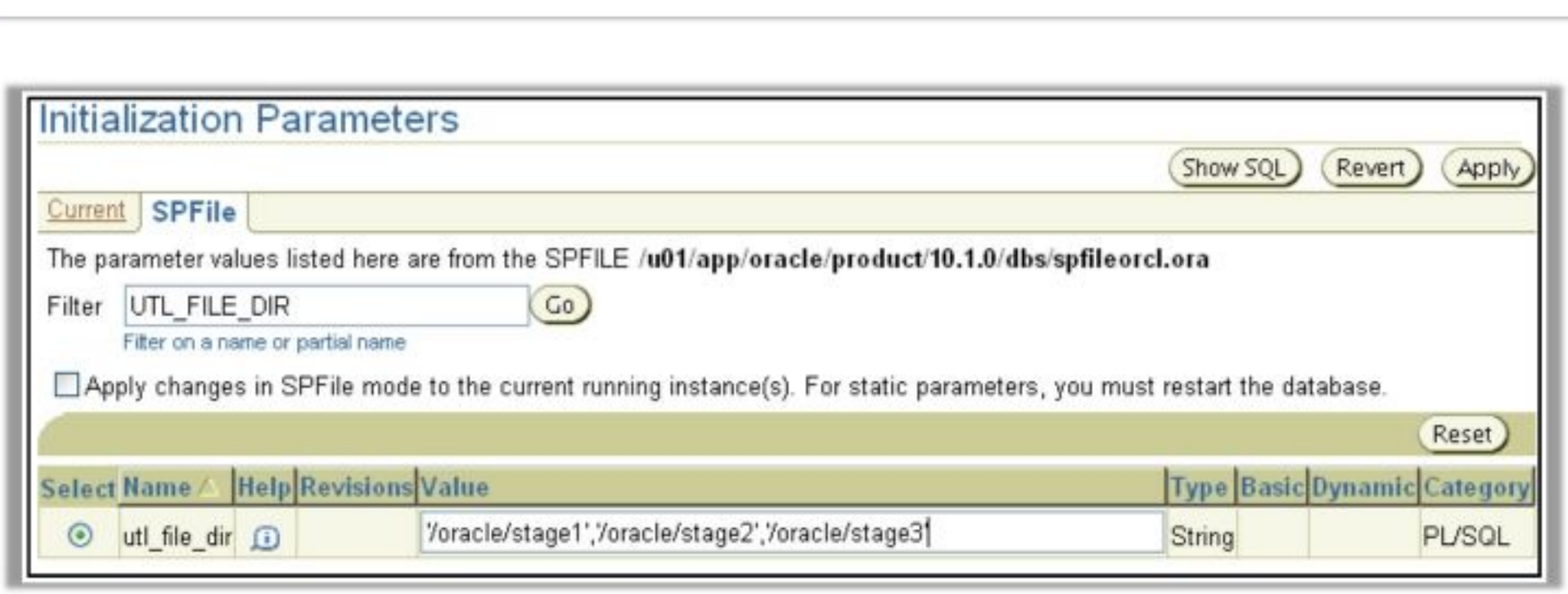
Securing Data

- **Enable Data Dictionary Protection**
  - To prevent users having ANY TABLE system privileges from modifying the content present in the data dictionary, enable **data dictionary protection**
  - Enabling data dictionary protection enables SYS users to login as **SYSDBA** only and no other user
  - Data dictionary protection is enabled by default
  - To enable data dictionary protection ensure that O7_DICTIONARY_ACCESSIBILITY is FALSE
  - Set **O7_DICTIONARY_ACCESSIBILITY = FALSE** in initsid.ora control file or set O7_DICTIONARY_ACCESSIBILITY = FALSE in a server parameter file
- **Limit operating system access**
  - Make sure to restrict user access to data files, log files, trace files, external tables, BFILE data types
- **Encrypt sensitive data along with backup media containing database files**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Restrict Access to Operating System Directories

- Configure the **UTL_FILE_DIR** parameter to :
  - List the directories available for PL/SQL file I/O
  - List database server directories to which the user has read and write permissions



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Securing Database Installation and Configuration

- To **prevent users from overwriting** installation files, installation should be done offline. If installation is done on the network make sure to disconnect logged in users before installation
- To **reduce the attack surface**, install only required components and remove components which will not be used in the production server
- Use the latest version and patches. Outdated and unpatched software increases security risks. Browse the Oracle website for the latest patches and versions
- Make sure to set the "unmask" value to 022 before beginning installation on a UNIX system. Improper "unmask" settings result in unrestricted permissions
- Check the permissions of important folders and files created while installation to prevent unauthorized access
- **Example :**
  - Set the permission to 0750 or less for the files present in **$ORACLE_HOME except $ORACLE_HOME/bin**
  - Set the permission to 0755 or less for the files present in **$ORACLE_HOME/bin**
- To prevent **sysdba login** to the database ensure that only the Oracle software owner belongs to dba
- Ensure that the "Tkprof" utility is either removed or disabled from production. "Tkprof" utility formats SQL trace output to human readable format. This becomes a source for attackers to identify issues in running the database

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Securing Network

- Ensure that the **listener** is secured
  - Set the listener password
  - Turn on logging
  - Set ADMIN_RESTRICTIONS in listener.ora to disable all runtime modifications
  - Set a different listener name than LISTENER
  - Set directories for tracing
  - Use IP addresses rather than hostnames
  - Ensure that the listener password is not stored in listener.ora file
- Create separate listeners for clients and for administration
- Ensure the use of a **firewall** to protect **Oracle** (Windows)
- Accept connections from short list of **IP addresses**
- View the **sqlnet.log** files on the server and client machines
- Check with the port scanner for open default ports
- Secure the intelligent agent
- Ensure that the connection is configured to implement encryption between clients and the database

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

How to Configure Encryption on the Client and the Server

- Open **Net Manager**
- In **Windows**: Navigate **Start→ Programs→ Oracle - HOME_NAME→ Configuration and Migration Tools→Net Manager**
- Select **Profile** from **Oracle Net Configuration → Local→ Profile**
- Select **Network Security** from **Naming List** to open **Network Security** window
- Click the **Encryption** tab
- For the Encryption option select either **Client** or **Server**
- Select one of the following as **Encryption Type (REQUESTED, REQUIRED, ACCEPTED, REJECTED)**
- Enter **Encryption Seed** of random characters of 10 -70 length (Client encryption should not match with the server encryption seed)
- Select one of the listed algorithms
  - Navigate to **File→ Save Network Configuration to update the sqlnet.ora** file



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Control Access Data

- Oracle has various methods to handle data securely. The following are some of the **methods** implemented to protect data:
  - **Oracle Virtual Private Database:** It creates a policy which enforces a **WHERE** clause for all SQL statements and restricts access to row and column level data
  - **Oracle Label Security:** This feature provides restriction for access to the database table at the **row level**
  - **Oracle Database Vault:** This feature restricts the privileged users and **DBA** of environment

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Virtual Private Database

- **Virtual Private Database** helps DBA to define and implement row-level access control policies based on session attributes
- When a user accesses a table, view or synonym having a **VPD policy** attached
- Oracle server executes the attached policy function when ever a user executes a query
- Considering the **session attribute** or contents of the database, the policy function returns a predicate which is concatenated as a where clause
- The server executes the altered **SQL query**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

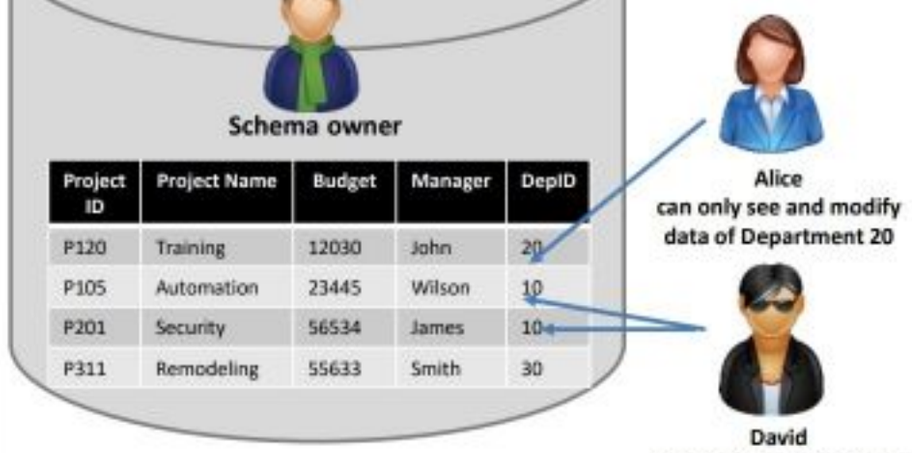
Virtual Private Database (Cont'd)

- **Implementation Virtual Private Database:**
  - **Step 1 : Create a policy function**

```
Create function sec_function(p_schema varchar2, p_obj varchar2)
Return varchar2 As  user VARCHAR2(100);
Begin
if ( SYS_CONTEXT('userenv', 'ISDBA') ) then return '';
//Admin can access any data
else
user := SYS_CONTEXT('userenv', 'SESSION_USER'); return 'owner = ' || user;
// Users can only access their own data end if; End;
```

  - **Step 2 : Link created policy to a table**

```
execute dbms_rls.add_policy
(object_schema => 'Alice', object_name => 'my_table',policy_name => 'my_policy',
function_schema => 'Alice', policy_function => 'sec_function', statement_types =>
select, update, insert', update_check => TRUE );
```



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Oracle Label Security

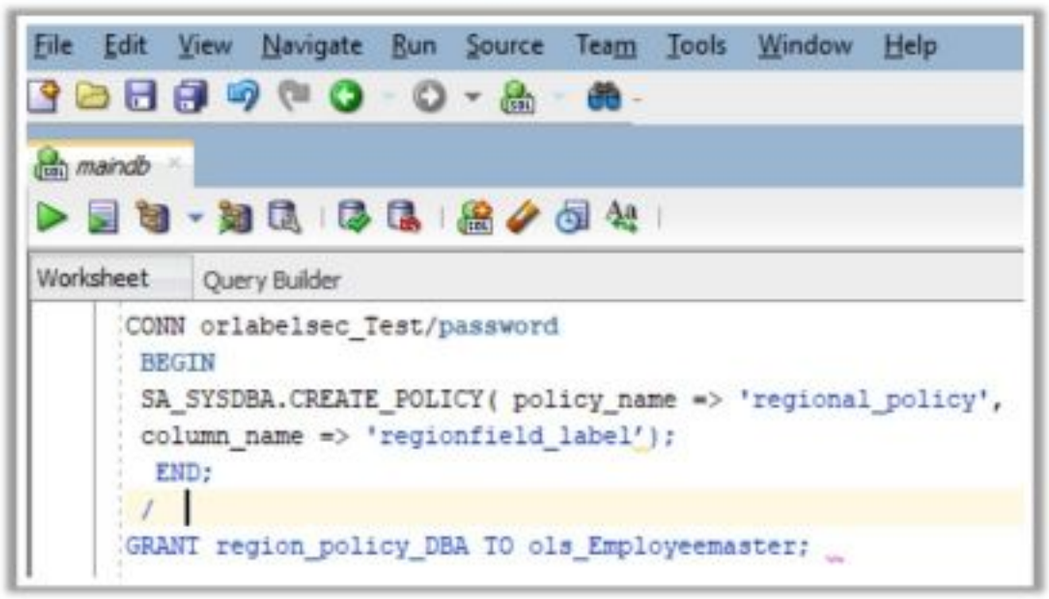
- **Oracle Virtual Private Database** technology implements **Oracle label security**. Each row having labels it compares with user label and privileges before displaying

Implementation of Oracle Label Security:

- To implement Oracle label security user has to checkout following steps:

Step1	Installation Oracle Label Security
Step2	Configure Instance
Step3	Create Policy to specify column
Step4	Define Component of label
Step5	Create Schema
Step6	Create Label Function
Step7	Apply Policy to table
Step8	Initialize Label
Step9	Test The Label Security Policy

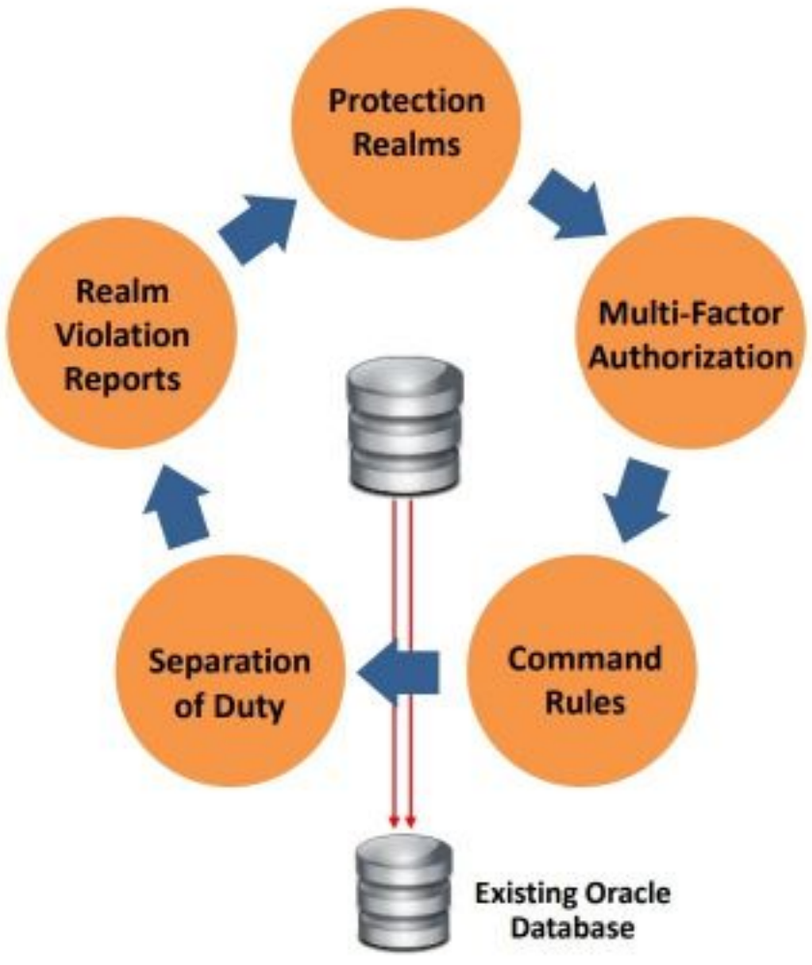
Creating Policy for Test User



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Database Vault

- Database vault protects the database against insider attacks by implementing **realms, factors, and command rules**
- Implements restriction on privileged users by:
  - Restricting privileged users from accessing application data
  - Enforcing separation of duty
- Implements Real time access controls by:
  - Restricting access based on IP address, authentication method, time of day etc.



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

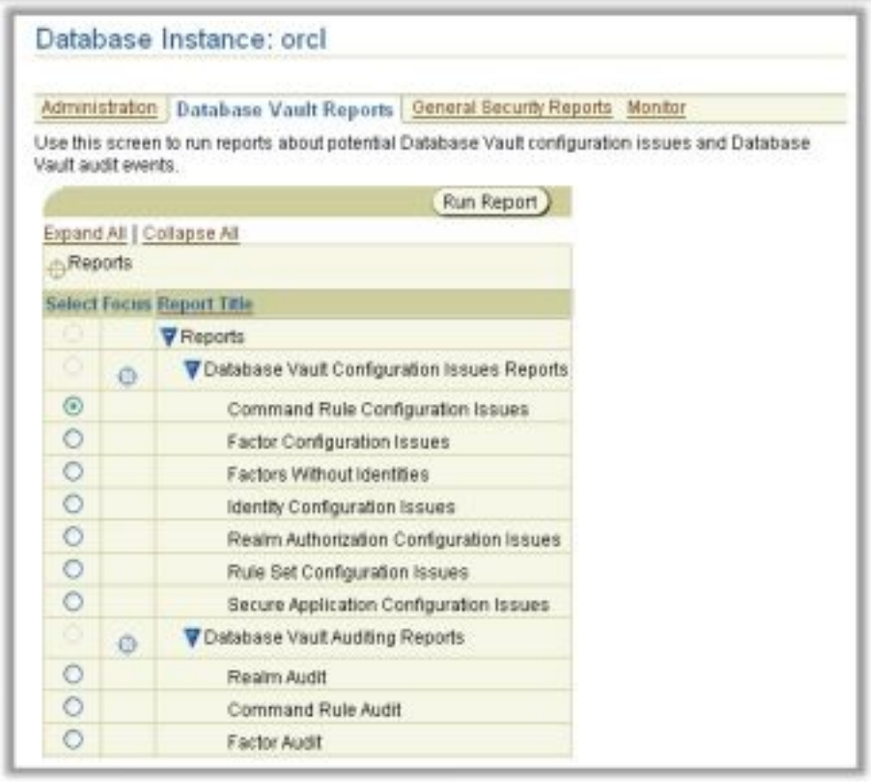
Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Database Vault: Management and Reports

- **Database Vault Administrative Interface** contains web based Administrative Interface to manage **Realms, Rules, Factors, Reports, Dashboard**
- Database Vault Reporting
  - Generates various security reports for compliance
  - Generates Audit violation attempts
  - Generates Realm, Rule and Factor Reports
  - Generates System and Public Privileges reports



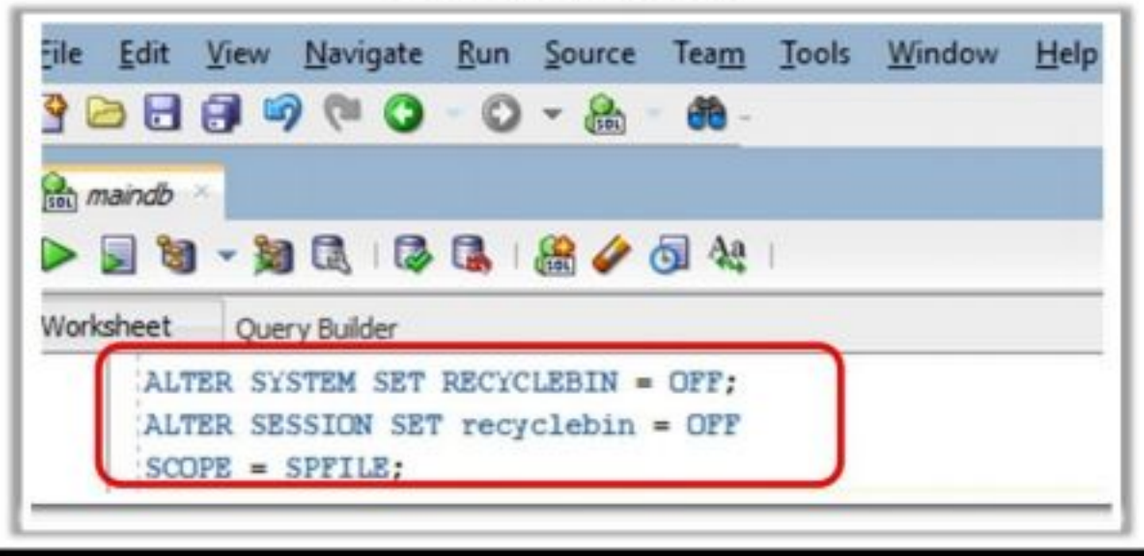


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Database Vault: Disabling the Recycle Bin

- To prevent dropped database objects from exposure, disable the recycle bin
- SYS users having SYSDBA administrative privileges or users with **ALTER SYSTEM** system privilege can disable Recycle Bin

To Disable Recycle Bin



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Audit Vault

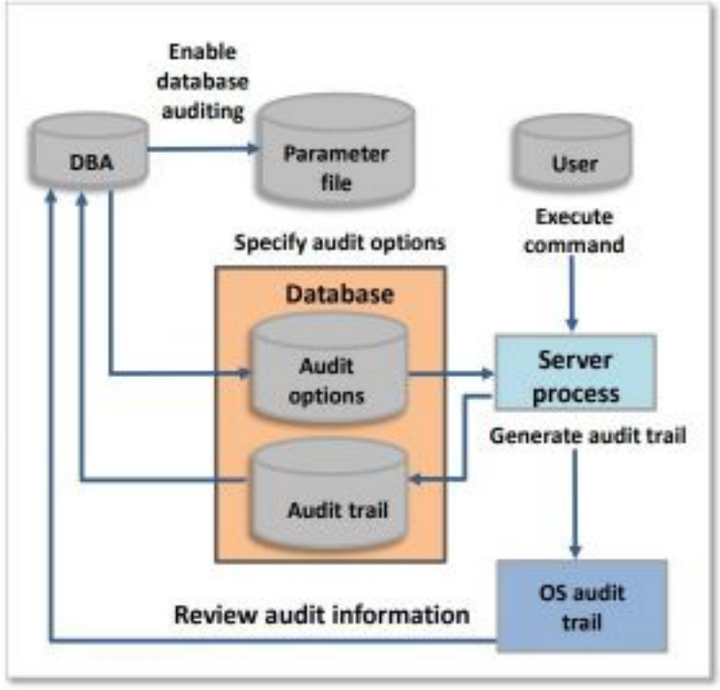
Audit Vault : Automated **Activity Management and Audit reporting**

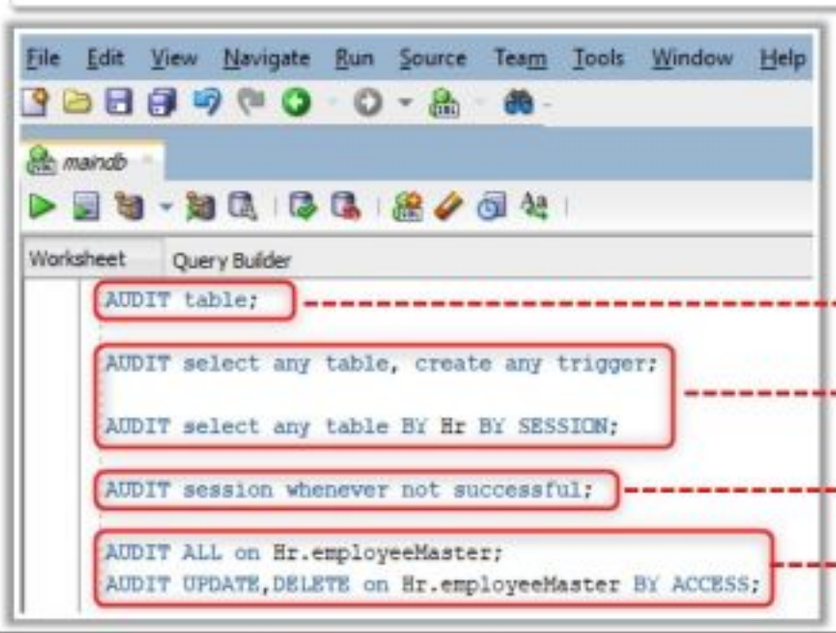
1. Centralized audit policy management
2. Consolidates audit data into a secure repository
3. Identifies and alerts on detecting suspicious activities
4. Compliance reporting

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Built-in Audit Tools: Standard Database Auditing

- Audits for privileged use including object access
- Can audit login events, use of system privileges, use of object privileges, SQL statements used
- Use **AUDIT_TRAIL** parameter to enable standard database auditing
  - NONE: Disable audit records
  - DB: Enable audit of records stored in the database
  - OS: Enable audit of records stored in the operating system audit trail





Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Built-in Audit Tools: Standard Database Auditing (Cont'd)

Viewing Auditing Options

Data Dictionary View	Description
ALL_DEF_AUDIT_OPTS	Default audit options
DBA_STMT_AUDIT_OPTS	Statement auditing options
DBA_PRIV_AUDIT_OPTS	Privileged auditing options
DBA_OBJ_AUDIT_OPTS	Schema object auditing options

Viewing Auditing Results

Audit Trail View	Description
DBA_AUDIT_TRAIL	All audit trail entries
DBA_AUDIT_EXISTS	Records for AUDIT EXIST/ DO NOT EXISTS
DBA_AUDIT_OBJECT	Records concerning schema objects
DBA_AUDIT_SESSION	All connect and disconnect entries
DBA_AUDIT_STATEMENT	Statement auditing records

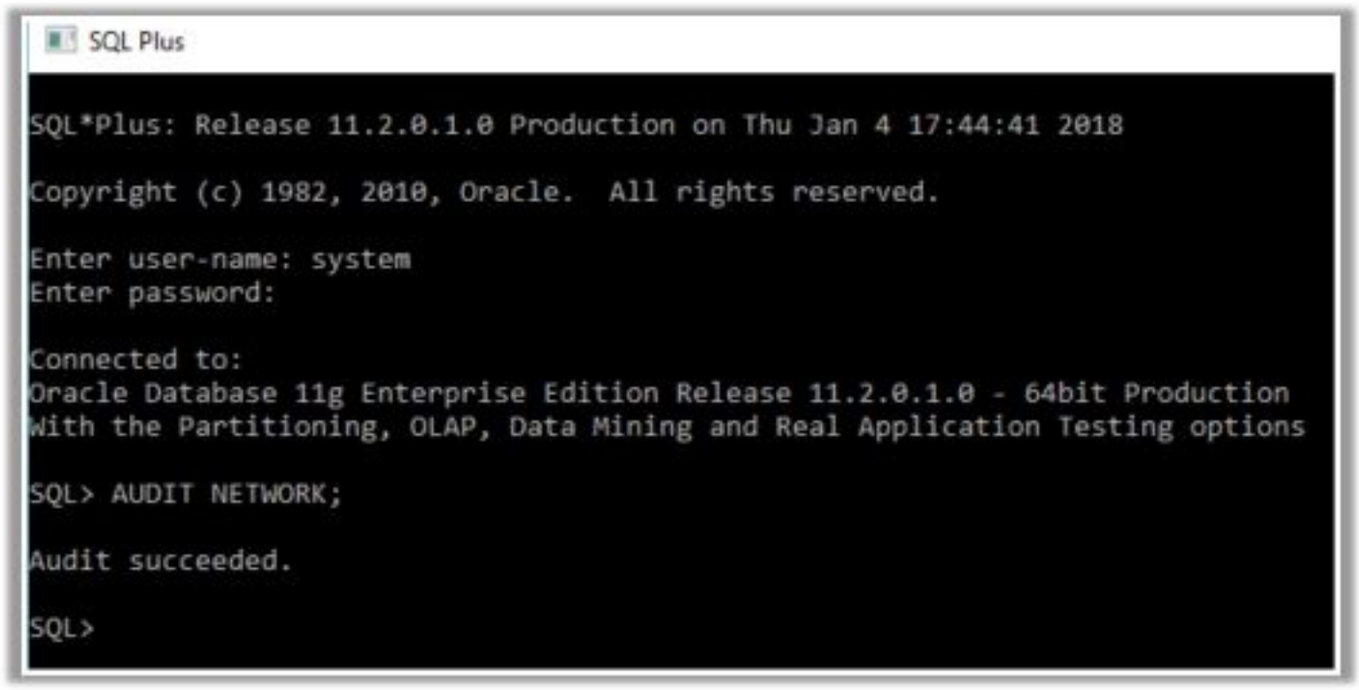
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Standard Auditing Enable Network Auditing

Follow the given steps to Enable Network Auditing

- Navigate to the **Start** menu in **Windows** and enter **SQLPLUS**
- Open **SQL*Plus Command Prompt** and use the command as shown here

```
SQL>Enter user-name: system
Enter password: SQL>AUDIT NETWORK;
```

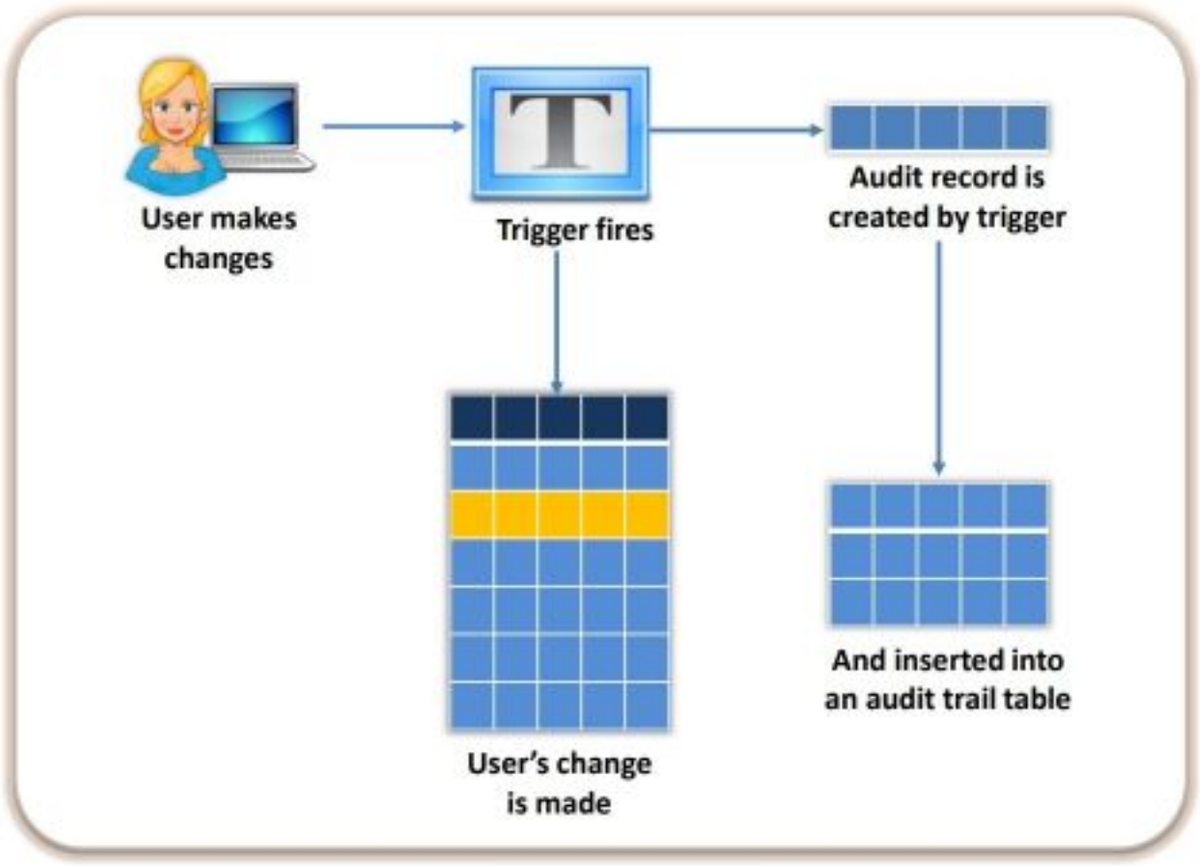


Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Built-in Audit Tools: Value Based Auditing

- Value-based auditing: **Audits data** changed by **DML statements**



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Built-in Audit Tools: Fine Grained Auditing (FGA)

- Audits SQL statements **(Insert, Update, Delete, and Select)**
- Audits data access based on content
- Can be implemented on a table or view
- Can execute a procedure
- Is administered with the **DBMS_FGA** package

DBMS_FGA package

Subprogram	Description
ADD_POLICY	Creates an audit policy using the supplied predicate as the audit condition
DROP_POLICY	Drops an audit policy
ENABLE_POLICY	Enables an audit policy
DISABLE_POLICY	Disables an audit policy

Data Dictionary Views

View Name	Description
DBA_FGA_AUDIT_TRAIL	All FGA events
ALL_AUDIT_POLICIES	All FGA policies for objects the current user can access
DBA_AUDIT_POLICIES	All FGA policies in the database
USER_AUDIT_POLICIES	All FGA policies for objects in the current user schema

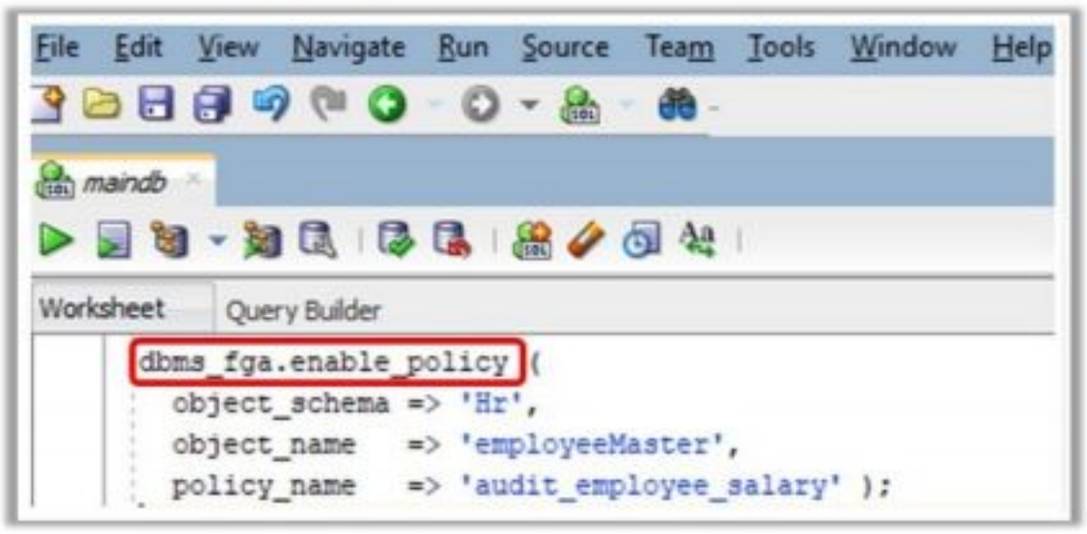
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

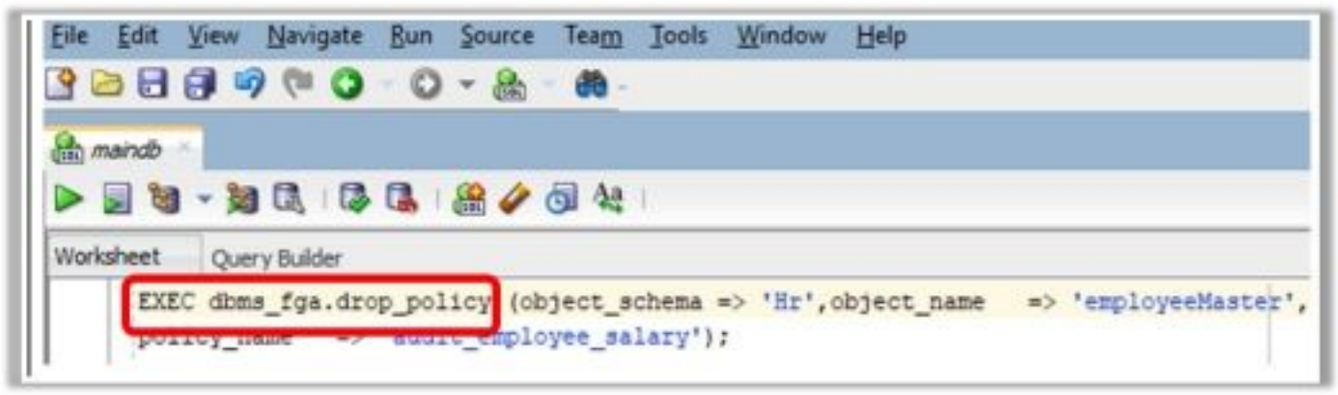
@ErrorCatchers

Built-in Audit Tools: Fine Grained Auditing (FGA) (Cont'd)

Enabling a FGA Policy



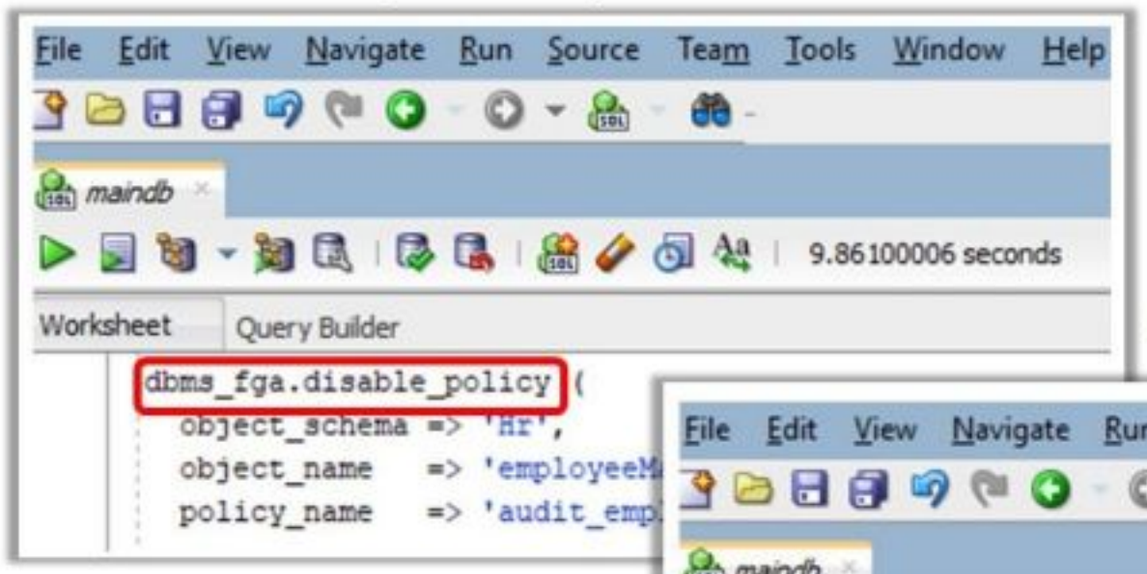
Dropping a FGA Policy



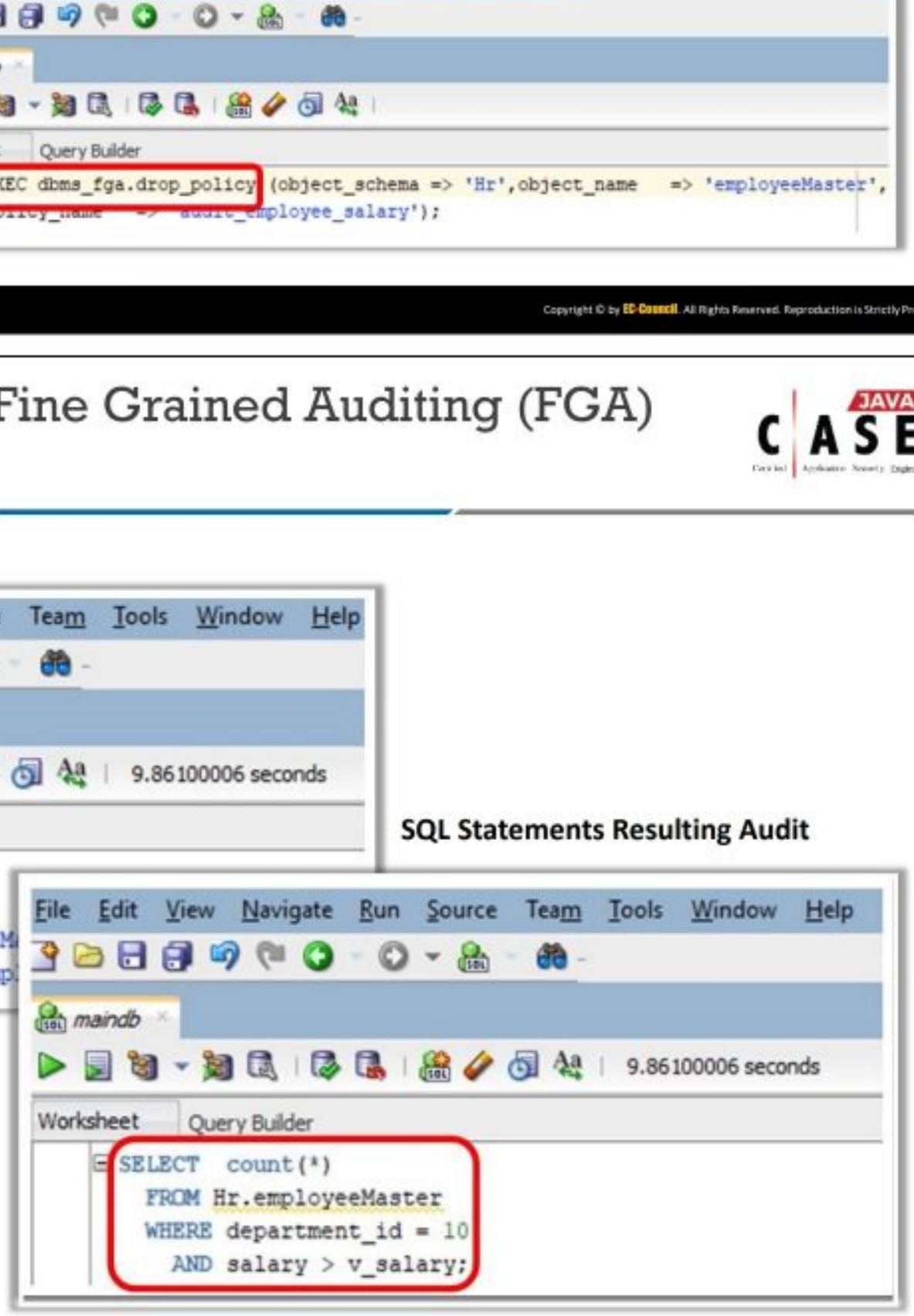
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Built-in Audit Tools: Fine Grained Auditing (FGA) (Cont'd)

Disabling a FGA Policy



SQL Statements Resulting Audit



Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

@ErrorCatchers

Recommended Audit Settings

- Ensure that **auditing** is **selective** and **effective**
- Auditing should focus on
  - Privileges, users
  - Selected tables with sensitive information
  - Secure configurations
- **Oracle provides following default audit configuration**

- ALTER ANY PROCEDURE
- CREATE ANY JOB
- DROP ANY TABLE
- ALTER ANY TABLE
- CREATE ANY LIBRARY
- DROP PROFILE
- ALTER DATABASE
- CREATE ANY PROCEDURE
- DROP USER
- ALTER PROFILE
- CREATE ANY TABLE
- EXEMPT ACCESS POLICY

- AUDIT ROLE BY ACCESS
- CREATE EXTERNAL JOB
- GRANT ANY OBJECT PRIVILEGE
- ALTER SYSTEM
- CREATE PUBLIC DATABASE LINK
- GRANT ANY PRIVILEGE
- ALTER USER
- CREATE SESSION
- GRANT ANY ROLE
- AUDIT SYSTEM
- CREATE USER
- AUDIT SYSTEM BY ACCESS
- DROP ANY PROCEDURE

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Security Maintenance and Monitoring

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Post Deployment Activities: Security Maintenance and Monitoring

1. Maintenance and monitoring is **iterative process** undertaken after initial deployment of the application
2. It includes set of activities that are being carried out to continuously maintain the security of web hosting environment. These activities allows to keep application up-to-date concerning the emerging vulnerabilities
3. The maintenance activities should be carried out at various level of web hosting environment
   - **OS level**
   - **Web server level**
   - **Application level**
4. The purpose of implementing the maintenance should be to modify the product without affecting its **integrity**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Security Maintenance Activities at OS Level

1. Maintain **Test server** and **Production server** separately
2. Look for the latest OS level security updates, patches, and hotfixes released continuously and apply them time to time
3. Monitor and analyze **system level logs**
4. Take **backup** of the data and OS regularly
5. Monitor the antivirus software to ensure updates are applied and functioning properly
6. Ensure **OS permissions** of all system folders are intact
7. Scan and update the system with the latest **antivirus** scan engine/virus definitions regularly
8. Ensure that spam/hacked services are not present by inspecting default start-up state of system services
9. Monitor **background processes** and startup items to prevent from malware infections continuously

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Security Maintenance Activities at Web Container Level

I. Continuously look for the latest **versions, security updates, patches** and apply them time to time

II. Monitor and analyze Web container **logs**

III. Ensure that configured **web container security features** are intact

IV. Scan the **web server** periodically for identifying vulnerabilities and misconfigurations

V. Perform the penetration testing periodically to assess the effectiveness of existing security features

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Security Maintenance Activities at Application Level

- Scan the application periodically for identifying **vulnerabilities** and **misconfigurations**
- Perform the **penetration testing** periodically to assess the effectiveness of existing security features
- **Patch** the application for vulnerabilities discovered
- Maintain **backup** of public websites

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Module Summary

- Security should be the critical consideration while deploying any application
- JAVA web application secure deployment involves ensuring security at various levels from bottom to top
- Administrator should ensure the physical security of a host machine, its OS security, and security of the all other software installed on the machine
- A Web Application Firewall (WAF) provides a security layer that protects the web server from malicious traffic
- Administrator should ensure secure setting of the web server (Apache Tomcat, Jboss(WildFly))
- Administrator should configure and check the deployment security settings in both Server.xml and web.xml files carefully
- Maintenance and monitoring is an iterative process undertaken after the initial deployment of the application

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers

References

Module 01: Understanding Application Security, Threats, and Attacks

1. J.D. Meier, Alex Mackman, Blaine Wastell, Cheat Sheet: Web Application Security Frame, last modified: 2005, http://msdn.microsoft.com/en-us/library/ff649461.aspx.
2. Ruby Qurashi, Eight steps for integrating security into application development, last modified: 2005, http://www.computerworld.com/s/article/106805/Eight_steps_for_integrating_security_into_application_development?taxonomyId=17&pageNumber=1.
3. J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla, Anandha Murukan, Threat Modeling, last modified: 2003, http://msdn.microsoft.com/en-us/library/ff648644.aspx.
4. Russ McMahon, Security Development Lifecycle, accessed: December 22, 2017, https://www.owasp.org/images/4/42/Microsoft-SDL2.pdf.
5. Security Testing a .NET Application, accessed: December 22, 2017, http://etutorials.org/Programming/Programming+.net+security/Part+I+Fundamentals/Chapter+4.+The+Lifetime+of+a+Secure+Application/4.3+Security+Testing+a+.NET+Application/.
6. Ong Khai Wei, Securing Your Web Application against security vulnerabilities, accessed: December 22, 2017, http://www-07.ibm.com/sg/smarterbusiness/meettheexperts/includes/downloads/Securing_Your_Web_0910_eve.pdf.
7. Mark Sherman, Building Resilient Systems: The Secure Software Development Lifecycle, accessed: December 22,2017, http://higherlogicdownload.s3.amazonaws.com/AUVSI/c2a3ac12-b178-4f9c-a654-78576a33e081/UploadedImages/Proceedings/Breakouts/Cybersecurity/Mark%20Sherman-CMU_150722%20Automated%20Vehicles%20Symposium%20-%20CYBERSECURITY%20FOR%20AUTOMATED%20VEHICLES%20Breakout%20-%20Sherman%20v4%20-%20distribution(2).pdf.
8. Lalit Kale, Application Security-Understanding The Horizon, last modified: 2014, https://www.slideshare.net/lalitkale/application-securityi-understanding-the-horizon.
9. John Colley, Why Secure Coding is not Enough: Professionals' Perspective, accessed: December 22, 2017, https://link.springer.com/chapter/10.1007%2F978-3-8348-9363-5_30#page-1.
10. J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla, Anandha Murukan, Web Application Security Methodology, accessed: December 22, 2017, http://www.guidanceshare.com/wiki/Web_Application_Security_Methodology.
11. File: Security in the SDLC Process.png, accessed: December 22, 2017, https://www.owasp.org/index.php/File:Security_in_the_SDLC_Process.png.
12. Software Security Maturity Models – BSIMM & OpenSAMM, last modified: 2016, https://jasonamorrow.com/bsimm-opensamm/.
13. State of Application Security Infographic, last modified: 2013, http://www.quotium.com/resources/state-of-application-security-infographic/.

Module 02: Security Requirements Gathering

14. Edward van Deursen, Jan Jaap Cannegieter, Eliciting security requirements needs a different process, accessed: January 2, 2018, https://re-magazine.ireb.org/issues/2015-2-bridging-the-impossible/eliciting-security-requirements/.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

15. WHAT ARE SECURITY REQUIREMENTS?, accessed: January 2, 2018, https://www.securitycompass.com/media/pdf/article-what-are-security-requirements.pdf.
16. Application Security by Design, accessed: January 2, 2018, https://web.securityinnovation.com/hs-fs/hub/49125/file-14368892-pdf/whitepapers/application_security_by_design.pdf.
17. Nancy R. Mead, Eric D. Hough, Theodore R. Stehney, Security Quality Requirements Engineering (SQUARE) Methodology, last modified: 2005, https://resources.sei.cmu.edu/asset_files/technicalreport/2005_005_001_14594.pdf.
18. Jose Romero-Mariona, Hadar Ziv, Debra J. Richardson, Security Requirements Engineering: A Survey, last modified: 2008, https://isr.uci.edu/tech_reports/UCI-ISR-08-2.pdf.
19. P.Salinia, S.Kanmani, Security Requirements Engineering Process for Web Applications, accessed: January 2, 2018, https://www.sciencedirect.com/science/article/pii/S1877705812022412.
20. Hester Fox, Developing Secure Software, accessed: January 2, 2018, http://slideplayer.com/slide/10911394/.
21. Misuse case, accessed: January 2, 2018, https://en.wikipedia.org/wiki/Misuse_case.
22. Paco Hope, Peter White, Software Security Requirements, accessed: January 2, 2018, http://sqgne.org/presentations/2007-08/Hope-Sep-2007.pdf.
23. Donald G. Firesmith, Security Use Cases, accessed: January 2, 2018, http://www.jot.fm/issues/issue_2003_05/column6/.
24. Joshua J. Pauli, Refining Use/Misuse/Mitigation Use Cases for Security Requirements, accessed: January 2, 2018, http://file.scirp.org/pdf/JSEA_2014070811065395.pdf.
25. Steven Thomas, Abuser Story – User Stories to Prevent Hacking, last modified: 2014, http://itsadeliverything.com/abuser-story-user-stories-to-prevent-hacking.
26. Jim Bird, Adding Appsec to Agile: Security Stories, Evil User Stories and Abuse(r) Stories, accessed: January 2, 2018, https://dzone.com/articles/adding-appsec-agile-security.
27. Annette Tetmeyer, A POS Tagging Approach to Capture Security Requirements within an Agile Software Development Process, accessed: January 2, 2018, https://kuscholarworks.ku.edu/bitstream/handle/1808/11464/Tetmeyer_ku_0099M_12820_DATA_1.pdf;sequence=1.
28. Chandramohan Muniraman, Meledath Damodaran, A PRACTICAL APPROACH TO INCLUDE SECURITY IN SOFTWARE DEVELOPMENT, accessed: January 2, 2018, http://www.iacis.org/iis/2007/Muniraman_Damodaran.pdf.
29. SAMM - Security Requirements – 1, accessed: January 2, 2018, https://www.owasp.org/index.php/SAMM_-_Security_Requirements_-_1.
30. Jamie Boote, Software Integrity, last modified: 2016, https://www.synopsys.com/blogs/software-security/software-security-requirement/.
31. FRANK RIETTA, What Is an Abuser Story (Software), last modified: 2015, https://rietta.com/blog/2015/10/11/what-is-an-abuser-story-software/.
32. Agile Software Development: Don't Forget EVIL User Stories, accessed: January 2, 2018, https://www.owasp.org/index.php/Agile_Software_Development:_Don%27t_Forget_EVIL_User_Stories .
33. Adam Shostack, Threat Modeling: Designing for Security, accessed: January 2, 2018, https://books.google.co.in/books?id=YiHcAgAAQBAJ&pg=PT620&lpg=PT620&dq=threat+trees&source=bl&ots=eSUlHC7_Oq&sig=vUjg0WxKM9VUvwQigjAJYeln940&hl=en&sa=X&ved=0ahUKEwjghvD2wOHQAhXDo5QKHR1uAbM4FBDoAQghMAE#v=onepage&q=threat%20trees&f=false.
34. E. Gottesdlener, Abuse Case Guidelines, accessed: January 2, 2018, https://www.ebgconsulting.com/Pubs/Articles/Abuse%20Case%20Guidelines.pdf.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

Module 03: Secure Application Design and Architecture

35. SECURE DESIGN AND ARCHITECTURE REVIEW, accessed: January 10, 2018, https://www.appsecurelabs.com/consulting/secure-design-and-architecture-review/#.
36. Application Security by Design, accessed: January 10, 2018, https://web.securityinnovation.com/hs-fs/hub/49125/file-14368892-pdf/whitepapers/application_security_by_design.pdf.
37. J.D. Meier, Alex Mackman, Blaine Wastell, How To: Create a Threat Model for a Web Application at Design Time, last modified: 2005, https://msdn.microsoft.com/en-us/library/ff647894.aspx.
38. Application Threat Modeling, accessed: January 10, 2018, https://www.owasp.org/index.php/Application_Threat_Modeling.
39. Attack Surface Analysis Cheat Sheet, accessed: January 10, 2018, https://www.owasp.org/index.php/Attack_Surface_Analysis_Cheat_Sheet.
40. Umut IŞIK, Threat modelling with sample application, last modified: 2014, https://www.slideshare.net/UmutIIK/threat-modelling-withsampleapplication.
41. Sherif Koussa, Simplified Security Code Review Process, last modified: 2013, https://www.slideshare.net/skoussa/simplified-security-code-review-process.
42. James Jasper Fletcher, Software Security Assessment, accessed: January 10, 2018, http://slideplayer.com/slide/5776271/.
43. Building Real Software, last modified: 2013, http://swreflections.blogspot.in/2013/06/what-is-important-in-secure-software.html.

Module 04: Secure Coding Practices for Input Validation

44. Open Source Validation Frameworks, accessed: January 18, 2018, http://java-source.net/open-source/validation.
45. Java validation filter, accessed: January 18, 2018, http://zetcode.com/java/validationfilter/.
46. Category:OWASP Enterprise Security API, accessed: January 18, 2018, https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API.
47. Data validations in Struts2, accessed: January 18, 2018, http://www.techmyguru.com/struts2/index.php?section=13/Data_validations_in_Struts2.
48. Validation, Data Binding, and Type Conversion, accessed: January 18, 2018, https://docs.spring.io/spring/docs/4.1.x/spring-framework-reference/html/validation.html.
49. Arvind Rai, Validation Framework in Spring with Example, last modified: 2013, https://www.concretepage.com/spring/validation-framework-spring-example.
50. Baeldung, Java Bean Validation Basics, last modified: 2018, http://www.baeldung.com/javax-validation.
51. Spring MVC form validation with custom validator, accessed: January 18, 2018, http://javainsimpleway.com/spring-mvc-form-validation-with-custom-validator/.

Module 05: Secure Coding Practices for Authentication and Authorization

52. RAJNISH BHATIA, ALL CODES, last modified: 2008, http://rajnishbhatia19.blogspot.in/search/label/Kerberos.
53. Willie Wheeler, SECTION 1 Introduction, accessed: January 25, 2018, https://dzone.com/refcardz/expression-based-authorization?chapter=1.
54. Spring Book – Chapter 15 – Web Application Security with Spring, last modified: 2013, https://www.javacodebook.com/2013/08/08/spring-book-chapter-15-web-application-security-with-spring/6/.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

55. Prasanth Gullapalli, Java Code Greeks, accessed: January 25, 2018, https://www.javacodegeeks.com/2013/11/spring-security-behind-the-scenes.html.

56. Alessandro, Spring Security Tutorial: Form Login, last modified: 2014, http://codehustler.org/blog/spring-security-tutorial-form-login/.

57. Jason Ferguson, Spring Security 3, last modified: 2011, https://www.slideshare.net/jasonferguson1/spring-security-3.

58. Willie Wheeler, SECTION 4 Web Authorization, accessed: January 25, 2018, https://dzone.com/refcardz/expression-based-authorization?chapter=4.

59. Willie Wheeler, SECTION 6 Web Authorization, Revisited, accessed: January 25, 2018, https://dzone.com/refcardz/expression-based-authorization?chapter=6.

60. Brad Rubin, Java security, Part 1: Crypto basics, last modified: 2002, https://www.ibm.com/developerworks/java/tutorials/j-sec1/j-sec1.html.

61. Eugen Paraschiv, Spring Security – Persistent Remember Me, last modified: 2016, http://www.baeldung.com/spring-security-persistent-remember-me.

62. Eugen Paraschiv, Spring Security Remember Me, last modified: 2016, http://www.baeldung.com/spring-security-remember-me.

63. Saeidzebardast, How to set Unlimited Session Timeout in Tomcat, last modified: 2016, https://coderwall.com/p/eocuyw/how-to-set-unlimited-session-timeout-in-tomcat.

64. Fusion Middleware Programming Security for Oracle WebLogic Server, accessed: January 25, 2018, https://docs.oracle.com/cd/E23943_01/web.1111/e13711/thin_client.htm#SCPRG133.

Module 06: Secure Coding Practices for Cryptography

65. Mkyong, Spring Security password hashing example, last modified: 2014, https://www.mkyong.com/spring-security/spring-security-password-hashing-example/.

66. Aaron Zauner, Javascript Object Signing & Encryption, last modified: 2015, https://www.slideshare.net/a_z_e_t/javascript-object-signing-encryption.

67. Dennis Detering, On the (in-)security of JavaScript Object Signing and Encryption, accessed: February 2, 2018, https://www.nds.rub.de/media/nds/arbeiten/2017/05/07/Masterthesis_Security-of-JOSE_Detering-FINAL.pdf.

68. Scott Arciszewski, Paseto is a Secure Alternative to the JOSE Standards (JWT, etc.), accessed: February 2, 2018, https://paragonie.com/blog/2017/03/jwt-json-web-tokens-is-bad-standard-that-everyone-should-avoid.

69. Brian Campbell, JOSE Can You See, last modified: 2014, https://www.slideshare.net/briandavidcampbell/jose-can-you-see-34360871.

70. Prabath Siriwardena, JWT, JWS and JWE for Not So Dummies! (Part I), last modified: 2016, https://medium.facilelogin.com/jwt-jws-and-jwe-for-not-so-dummies-b63310d201a3.

71. Javascript Object Signing and Encryption (JOSE), accessed: February 2, 2018, http://jose.readthedocs.io/en/latest/.

Module 07: Secure Coding Practices for Session Management

72. PANKAJ, Spring MVC Exception Handling – @ControllerAdvice, @ExceptionHandler, HandlerExceptionResolver, accessed: February 26, 2018, https://www.journaldev.com/2651/spring-mvc-exception-handling-controlleradvice-exceptionhandler-handlerexceptionresolver.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

73. Spring MVC Exception Handling, last modified: 2016, https://memorynotfound.com/spring-mvc-exception-handling/.
74. Exception Handling in Spring MVC with Example, mast modified: 2016, https://www.dineshonjava.com/exception-handling-in-spring-mvc/.
75. How to handle exceptions in Struts2, last modified: 2015, http://www.codejava.net/frameworks/struts/how-to-handle-exceptions-in-struts2.

Module 08: Secure Coding Practices for Error Handling

1. What is HttpSession?, accessed: February 15, 2018, https://www.studytonight.com/servlet/httpsession.php.
2. Eugen Paraschiv, Java Session Timeout, accessed: February 15, 2018, http://www.baeldung.com/servlet-session-timeout.
3. Johnmelton, Session Cookie HttpOnly Flag Java, last modified: 2012, https://www.whitehatsec.com/blog/session-cookie-httponly-flag-java/.
4. Managing Session Attributes For Mvc Applications, accessed: February 15, 2018, https://wastemanagement.site/info/spring-mvc-session-management-tutorial/managing-session-attributes-for-mvc-applications/ZbPmeb3HYqk.html.
5. Biju Kunjummen, Using Http Session With Spring Based Web Applications, last modified: 2014, https://dzone.com/articles/using-http-session-spring.
6. How to implement a HTTPS login page in a web application?, accessed: February 15, 2018, https://stackoverflow.com/questions/1454021/how-to-implement-a-https-login-page-in-a-web-application.
7. Secure Session Management Tips, last modified: 2011, https://wblinks.com/notes/secure-session-management-tips/.

Module 09: Static and Dynamic Application Security Testing (SAST & DAST)

8. CRV2 FrameworkSpecIssuesASPTop10, accessed: March 12, 2018, https://www.owasp.org/index.php/CRV2_FrameworkSpecIssuesASPTop10.
9. Secure Code Review, accessed: March 12, 2018, https://www.mitre.org/publications/systems-engineering-guide/enterprise-engineering/systems-engineering-for-mission-assurance/secure-code-review.
10. SECURE CODE REVIEW, accessed: March 12, 2018, https://www.appsecurelabs.com/consulting/secure-code-review/.
11. Naga Venkata Sunil Alamuri, Null meet Code Review, last modified: 2015, https://www.slideshare.net/sunilanv/null-meet-codereviewslideshare-5?qid=e80636f8-add5-4275-83a6-d3b03d46deee&v=&b=&from_search=3.
12. Secure Code Review: A Practical Approach, last modified: 2013, http://resources.infosecinstitute.com/secure-code-review-practical-approach/#gref.
13. secure_code_review_20141217.pptx, accessed: March 12, 2018, http://opensecuritytraining.info/SecureCodeReview_files/secure_code_review_20141217.pdf.
14. Security Code Review, accessed: March 12, 2018, https://www.torridnetworks.com/services/assessment/security-code-review.
15. Uncover Hidden Vulnerabilities With Security Code Review, accessed: March 12, 2018, https://www.paladion.net/source-code-review-services.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.
@ErrorCatchers

16. Security Code Review, last modified: 2013, https://www.youtube.com/watch?v=Q5ifAbg3ins&t=1712s.
17. Code Review, accessed: March 12, 2018, http://www.net-square.com/code_review.html.
18. J.D. Meier, Alex Mackman, Blaine Wastell, Prashant Bansode, Jason Taylor, Rudolph Araujo, How To: Perform a Security Code Review for Managed Code (.NET Framework 2.0), last modified: 2005, https://msdn.microsoft.com/en-us/library/ff649315.aspx.
19. Naga Venkata Sunil Alamuri, Null meet Code Review, last modified: 2015, https://www.slideshare.net/sunilanv/null-meet-codereviewslideshare-5?qid=e80636f8-add5-4275-83a6-d3b03d46deee&v=&b=&from_search=3.
20. Test, 1, 2, 3, test, test, last modified: 2011, https://www.64k-tec.de/2011/03/test-1-2-3-test-test/.
21. Source Code Analysis Tools, accessed: March 12, 2018, https://www.owasp.org/index.php/Source_Code_Analysis_Tools.
22. List of tools for static code analysis, accessed: March 12, 2018, https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis#.NET.
23. Eoin Keary, OWASP_AlphaRelease_CodeReviewGuide2.0.pdf, accessed: March 12, 2018, https://www.owasp.org/images/7/78/OWASP_AlphaRelease_CodeReviewGuide2.0.pdf.
24. OWASP CODE REVIEW GUIDE, accessed: March 12, 2018, https://www.owasp.org/images/2/2e/OWASP_Code_Review_Guide-V1_1.pdf.
25. Unvalidated Redirects and Forwards Cheat Sheet, accessed: March 12, 2018, https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.
26. Top 10 2010-A10-Unvalidated Redirects and Forwards, accessed: March 12, 2018, https://www.owasp.org/index.php/Top_10_2010-A10-Unvalidated_Redirects_and_Forwards.
27. Codereview-Authentication, accessed: March 12, 2018, https://www.owasp.org/index.php/Codereview-Authentication.
28. Philippe Arteau, Modern Static Analysis for .NET, last modified: 2016, http://gosecure.net/2016/11/23/modern-static-analysis-net/.
29. CWE-259: Use of Hard-coded Password, accessed: March 12, 2018, https://cwe.mitre.org/data/definitions/259.html.
30. CWE-321: Use of Hard-coded Cryptographic Key, accessed: March 12, 2018, https://cwe.mitre.org/data/definitions/321.html.
31. Use of hard-coded password, accessed: March 12, 2018, https://www.owasp.org/index.php/Use_of_hard-coded_password.
32. Use of hard-coded cryptographic key, accessed: March 12, 2018, https://www.owasp.org/index.php/Use_of_hard-coded_cryptographic_key.
33. Password Management: Hardcoded Password, accessed: March 12, 2018, https://www.owasp.org/index.php/Password_Management:_Hardcoded_Password.
34. Yamin Khakhu, Encrypt ConnectionString in Web.Config, last modified: 2014, https://www.codeproject.com/Tips/795135/Encrypt-ConnectionString-in-Web-Config.
35. How to: Secure Connection Strings When Using Data Source Controls, accessed: March 12, 2018, https://msdn.microsoft.com/en-IN/library/dx0f3cf2(v=vs.85).aspx.
36. Role-based authorization in ASP.NET Core, accessed: March 12, 2018, https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-2.1.
37. ASP.NET Authorization, accessed: March 12, 2018, https://msdn.microsoft.com/en-us/library/wce3kxhd.aspx.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

38. Gsark, "Your Login Attempt was not Successful. Please Try Again." - ASP.NET Login Control, last modified: 2009, https://www.codeproject.com/Articles/27682/Your-Login-Attempt-was-not-Successful-Please-Try.
39. ASP.NET 2.0 Security Questions and Answers – Authentication, accessed: March 12, 2018, http://www.guidanceshare.com/wiki/ASP.NET_2.0_Security_Questions_and_Answers_-_Authentication.
40. Sample Secure Code Review Report, accessed: March 12, 2018, https://www.mitre.org/sites/default/files/publications/secure-code-review-report-sample.pdf.
41. Anita D'Amico, Dynamic Application Security Testing Tools: Searching the Black Box, last modified: 2015, https://codedx.com/2015/01/30/dynamic-application-security-testing-tools/.
42. HARI BALASUNDARAM, A Baseline Approach to Security Testing, last modified: 2014, https://blog.box.com/blog/a-baseline-approach-to-security-testing/.
43. Kevin Fealey, What Good is this Tool? A Guide to Choosing the Right Application Security Testing Tools, last modified: 2015, https://www.slideshare.net/kfealey/what-good-is-this-tool-final.
44. Apoorva Phadke, AST vs. DAST: What's the best method for application security testing?, last modified: 2016, https://www.synopsys.com/blogs/software-security/sast-vs-dast/.
45. Learn About WebInspect Automated Dynamic Scans (DAST), last modified: 2015, https://www.youtube.com/watch?v=j0idyQrkPGE.
46. Brian Shura , Web Application Security Scanner List, accessed: http://projects.webappsec.org/w/page/13246988/Web%20Application%20Security%20Scanner%20List .
47. Category:Vulnerability Scanning Tools, accessed: March 12, 2018, https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools.
48. Using Burp to Detect SQL Injection Flaws, accessed: March 12, 2018, https://support.portswigger.net/customer/portal/articles/1965677-Methodology_SQL_Injection_.html.
49. Using Burp to Exploit SQL Injection Vulnerabilities: The UNION Operator, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/2163777-Methodology_SQL%20Exploitation_Union.html.
50. Using Burp to Brute Force a Login Page, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1964020-Methodology_Attacking%20Authentication_Brute%20Force%20Login.html.
51. Using Burp to Test for Missing Function Level Access Control, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1965720-Methodology_Missing%20Function%20Level%20Access%20Control.html#ForcedBrowsing.
52. Using Burp Scanner to Find Cross-Site Scripting (XSS) Issues, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1965737-Methodology_Using%20Burp%20Scanner%20To%20Find%20XSS.html.
53. Using Burp to Test for Insecure Direct Object References, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1965691-Methodology_Insecure%20Direct%20Object%20References.html.
54. Using Burp to Test for Security Misconfiguration Issues, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1965728-Methodology_Security%20Misconfiguration.html.
55. Using Burp to Test for Sensitive Data Exposure Issues, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1965730-Methodology_Sensitive%20Data%20Exposure.html.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

56. Using Burp's Site Map to Test for Access Control Issues, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1969844-Methodology_Attacking%20Access%20Controls_User%20Accounts.html.

57. Using Burp's "Request in Browser" Function to Test for Access Control Issues, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1969842-Methodology_Attacking%20Access%20Controls_Request%20in%20Browser.html.

58. Using Burp to Test for Cross-Site Request Forgery (CSRF), accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1965674-Methodology_Cross-Site%20Request%20Forgery%20(CSRF).html.

59. Using Burp to Test for Components with Known Vulnerabilities, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1965732-Methodology_Testing%20for%20Components%20with%20Known%20Vulnerabilities.html/

60. Using Burp to Test for Open Redirections, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1965733-Methodology_Testing%20for%20Open%20Redirections.html.

61. Using Burp to Bypass Hidden Form Fields, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1965741-Methodology_Bypassing%20Client-Side%20Controls_Hidden%20Form%20Fields.html.

62. Using Burp to Bypass Client Side JavaScript Validation, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1964212-Methodology_Bypassing%20Client-Side%20Controls_JavaScript%20Validation.html.

63. Using Burp to Hack Cookies and Manipulate Sessions, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1964073-Methodology_Attacking%20Session%20Management_Hacking%20Cookies.html.

64. Using Burp to Test Session Token Generation, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/1964169-using-burp-to-test-session-token-generation.

65. Using Burp Scanner to Test for DOM-Based, accessed: March 12, 2018, XSShttps://support.portswigger.net/customer/en/portal/articles/2325926-using-burp-scanner-to-test-for-dom-based-xss.

66. Using Burp to find Clickjacking Vulnerabilities, accessed: March 12, 2018, https://support.portswigger.net/customer/en/portal/articles/2363105-using-burp-to-find-clickjacking-vulnerabilities.

67. Gareth Heyes, Burp Clickbandit: A JavaScript based clickjacking PoC generator, last modified: 2015, https://portswigger.net/blog/burp-clickbandit-a-javascript-based-clickjacking-poc-generator.

68. Marcel Birkner, Automated Security Testing of web applications using OWASP Zed Attack Proxy, last modified: 2013, https://blog.codecentric.de/en/2013/10/automated-security-testing-web-applications-using-owasp-zed-attack-proxy/.

69. Geethu Alexander, ZAP Penetration Testing: A simple Tutorial to Detect Vulnerabilities, last modified: 2016, http://www.toobler.com/blog/zap-penetration-testing-simple-tutorial.

70. Jim Bird, Choosing Between a Penetration Test and a Secure Code Review, last modified: 2013, https://dzone.com/articles/choosing-between-penetration.

71. Sarah Vonnegut, 5 Best Practices for the Perfect Secure Code Review, last modified: 2016, https://www.checkmarx.com/2016/02/05/5-best-practices-perfect-secure-code-review/.

72. David Jacobs, Reviewing applications for security: Code review best practices, accessed: March 12, 2018, https://searchitchannel.techtarget.com/tip/Reviewing-applications-for-security-Code-review-best-practices.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

73. Secure Code Review, accessed: March 12, 2018, https://www.mitre.org/publications/systems-engineering-guide/enterprise-engineering/systems-engineering-for-mission-assurance/secure-code-review.

Module 10: Secure Deployment and Maintenance

74. Fusion Middleware Securing a Production Environment for Oracle WebLogic Server, accessed: March 27, 2018, https://docs.oracle.com/cd/E21764_01/web.1111/e13705/practices.htm#LOCKD124.

75. J.D. Meier, Alex Mackman, Blaine Wastell, Prashant Bansode, Andy Wigley, Kishore Gopalan, patterns & practices Security Deployment Review Index, last modified: 2005, https://msdn.microsoft.com/en-us/library/ff648510.aspx.

76. J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan, Chapter 22 :Deployment Review, accessed: March 27, 2018, https://msdn.microsoft.com/en-us/library/aa302438.aspx.

77. Tjylen Veselyj, Web Application Firewall (WAF) DAST/SAST combination, last modified: 2014, https://www.slideshare.net/TjylenVeselyj/web-application-firewall-waf-dastsast-combination-30531258.

78. Logan Kipp, Ask a Security Professional: Firewall Series – Part Two: Web Application Firewalls, last modified: 2016, https://wpdistrict.sitelock.com/blog/firewall-series-part2-waf/.

79. Web Application Firewall, accessed: March 27, 2018, http://tectonicsecurity.com/managed-security-services/web-application-firewall/.

80. Web Application Firewall, accessed: March 27, 2018, https://www.owasp.org/index.php/Web_Application_Firewall.

81. Web Application Firewall (WAF), accessed: March 27, 2018, https://f5.com/glossary/web-application-firewall.

82. Margaret Rouse, Web application firewall (WAF), accessed: March 27, 2018, https://searchsecurity.techtarget.com/definition/Web-application-firewall-WAF.

83. Category:OWASP Best Practices: Use of Web Application Firewalls, accessed: March 27, 2018, https://www.owasp.org/index.php/Category:OWASP_Best_Practices:_Use_of_Web_Application_Firewalls.

84. Brad Causey, Introduction to Web application firewalls in the enterprise, accessed: March 27, 2018, https://searchsecurity.techtarget.com/feature/Introduction-to-Web-application-firewalls-in-the-enterprise.

85. Why WAF Is Not Enough for Protecting Your Database, last modified: 2015, http://www.hexatier.com/why-web-application-firewalls-waf-only-cannot-protect-your-databases/.

86. Michael Thumann, The 5 Myths of Web Application Firewalls, last modified: 2012, https://insinuator.net/2012/04/the-5-myths-of-web-application-firewalls-2/.

87. Securing tomcat, accessed: March 27, 2018, https://www.owasp.org/index.php/Securing_tomcat.

88. 15 Ways To Secure Apache Tomcat 8, last modified: 2016, https://www.upguard.com/articles/15-ways-to-secure-apache-tomcat-8.

89. Mthomas, Best Practices for Securing Apache Tomcat 7, last modified: 2011, http://www.tomcatexpert.com/blog/2011/11/02/best-practices-securing-apache-tomcat-7.

90. Tomcat Security, accessed: March 27, 2018, https://www.unidata.ucar.edu/software/thredds/current/tds/reference/TomcatSecurity.html.

91. Jason Brittain, Ian F. Darwin,Top, Ten Tomcat Configuration Tips, last modified: 2003, http://www.onjava.com/pub/a/onjava/2003/06/25/tomcat_tips.html?page=1.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

92. Apache Tomcat 7, accessed: March 27, 2018, https://tomcat.apache.org/tomcat-7.0-doc/config/filter.html#CSRF_Prevention_Filter.
93. Apache Tomcat 8, accessed: March 27, 2018, https://tomcat.apache.org/tomcat-8.0-doc/ssl-howto.html.
94. Oracle Database Hardening Guide, last modified: 2014, https://oracledock.wordpress.com/2014/10/15/oracle-database-hardening-guide/.
95. Hardening the Oracle 11g Database - Initial Steps, last modified: 2007, http://www.securedba.com/securedba/2007/12/hardening-the-o.html.
96. Authentication Methods, accessed: March 27, 2018, https://docs.oracle.com/cd/B19306_01/network.102/b14266/authmeth.htm#BABCGGEB.
97. Storing Passwords in an Oracle Database, accessed: March 27, 2018, https://oracle-base.com/articles/9i/storing-passwords-in-the-database-9i.
98. Oracle 11g Tutorial, accessed: March 27, 2018, http://eoracle11g.blogspot.in/p/alter-user-information.html.
99. Oracle Database 2 Day + Security Guide, accessed: March 27, 2018, https://docs.oracle.com/cd/E11882_01/server.112/e10575.pdf.
100. Configuring Oracle Database Network Encryption and Data Integrity, accessed: March 27, 2018, https://docs.oracle.com/database/121/DBSEG/asoconfg.htm#DBSEG020.
101. Securing the Network, accessed: March 27, 2018, https://docs.oracle.com/cd/B28359_01/server.111/b28337/tdpsg_network_secure.htm#TDPSG90000.
102. Oracle Label Security, accessed: March 27, 2018, https://oracle-base.com/articles/9i/oracle-label-security-9i#CreateTestSchema.

Certified Application Security Engineer Copyright © by **EC-Council**
All Rights Reserved.Reproduction is Strictly Prohibited.

@ErrorCatchers

This page is intentionally left blank.

@ErrorCatchers



EC-Council

EC-COUNCIL OFFICIAL CURRICULA

@ErrorCatchers