

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

```
<action-mappings>
  <action
    path="/docdownLoad"
    type="com.mysite.d2.action.DownloadAction" name="infoForm"
    scope="request"
    input=".download"
    validate="true"
  ></action>
</action-mappings>
```

Enable Struts Validator

Secure and Insecure Struts Validator Code



Vulnerable Code

```
<action-mappings>
  <action
    path="/docdownLoad"
    type="com.mysite.d2.action.DownloadAction" name="infoForm"
    scope="request"
    input=".download"
    validate="false"
  ></action>
</action-mappings>
```

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

Secure Code

```
<action-mappings>
  <action
    path="/docdownLoad"
    type="com.mysite.d2.action.DownloadAction" name="infoForm"
    scope="request"
    input=".download"
    validate="true"
  ></action>
</action-mappings>
```

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

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

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

Example : Non Field Validator

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

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

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

The screenshot shows a Java IDE interface with a Validator.xml file open in the editor. The XML code defines a visitor validator with parameters: fieldName set to 'user', context set to 'myContext', and appendPrefix set to 'true'. The code is as follows:

```
1 <validator type="visitor">
2   <param name="fieldName">user</param>
3   <param name="context">myContext</param>
4   <param name="appendPrefix">true</param>
5 </validator>
```

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 minlength 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

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()

```
File Edit Refactor Source Navigate Search Project Run Window Help
RegisterAction.java
1 package Validation;
2 public class Registration 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        if(password.length()<1)
12        {
13            addFieldError("password","Password must be greater than 5");
14        }
15    }
16 }
```

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

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 form
- 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

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

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

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

```
File Edit Source Refactor Navigate Search Project Run Window Help
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
11     @Override
12     public void validate(Object target, Errors errors) {
13         ValidationUtils.rejectIfEmpty(errors, "PersonName", "PersonName.empty");
14         employee prep=(employee)target;
15     }
16 }
```

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.POST)
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 user = (User) o;
18         ValidationUtils.rejectIfEmptyOrWhitespace(errors, "username", "NotEmpty");
19         if(user.getUsername().length() < 6 || user.getUsername().length() > 30) {
20             errors.rejectValue("username", "Size.userForm.username");
21         } if (userservice.findByUsername(user.getUsername()) != null) {
22             errors.rejectValue("username", "Duplicate.username");
23         }
24     }
25 }
```

- **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

Implementing Validator



- Validator can be implemented using `InvokeValidator()` method in the controller code

The screenshot shows the NetBeans IDE interface with the following details:

- Toolbar:** Includes icons for File, Edit, Refactor, Source, Navigate, Search, Project, Run, Window, Help, and various file operations like Open, Save, and Find.
- Quick Access:** A dropdown menu listing recent files: HomeController.java, UserValidator.java, UserService.java, and User.java.
- Code Editor:** Displays the Java code for `UserController.java`. The code implements a `UserValidator` and handles a POST request for user registration. A red box highlights the line of code where `ValidationUtils.invokeValidator(uservalidator, userform, errors)` is called.

```
File Edit Refactor Source Navigate Search Project Run Window Help
Quick Access
 HomeController.java UserValidator.java UserService.java User.java
 1 import org.springframework.ui.Model;
 2 import org.springframework.validation.BindingResult;
 3 import org.springframework.validation.Errors;
 4 import org.springframework.validation.ValidationUtils;
 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 import org.springframework.web.bind.annotation.RequestMappings;
10
11 @Controller
12 public class HomeController {
13
14
15     private UserValidator uservalidator;
16     @RequestMapping(value = "/UserRegistration", method = RequestMethod.POST)
17     public String registration(@ModelAttribute("userform") Userform userform, Bind
18     {
19         uservalidator.validate(userform, bindresult);
20
21         ValidationUtils.invokeValidator(uservalidator, userform, errors);
22
23
24
25         return "registration";
26     }
27 }
```

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

The screenshot shows the Eclipse IDE interface with several open tabs at the top: HomeControl..., UserValidator..., UserJava..., Test(junit4.xml), and a Quick Access search bar. The main editor area displays Java code for a class named HomeControl. A tooltip box is overlaid on the screen, highlighting the `getUsername` method. The tooltip contains the following information:

```
StartFrom  
private boolean working;  
@Size(min = 10, max = 200, message = "Enter user info and it must be  
private String aboutUser;  
@MinValue = 18, message = "User Age should more than 18")  
@MaxValue = 150, message = "User less than 150")  
private int age;  
public String getUsername() {  
    return null;  
}
```

Configuring JSR 380



Configure validator in the applicationContext

```
<?xml Version="1.0" encoding="UTF-8"?>
<bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
    <property name="validationMessageSource" ref="messageSource"/>
</bean>
```

*-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mvc:annotation-driven validator="validator" />
```

Spring MVC will automatically validates inputs from the controller

```
String userinfo(@Valid User user, Errors errors)
{
    if(errors.hasErrors())
    {
        return "adduser";
    }
    return "viewuser";
}
```

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

```
public class UserValidator implements Validator {
    @Override
    public boolean supports(Class<?> user) {
        return user instanceof User;
    }

    @Override
    public void validate(Object o, Errors errors) {
        User user = (User) o;
        int agerange = user.getAge();
        if(agerange < 18) {
            errors.rejectValue("age", "user.age.range.invalid");
        }
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "username", "NotEmpty");
        if(user.getUsername().length() < 6 || user.getUsername().length() > 30) {
            errors.rejectValue("username", "size.user.username");
        } if(userService.findByUsername(user.getUsername()) != null) {
            errors.rejectValue("username", "Duplicate.username");
        }
    }
}
```

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

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.

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: 30
30 import java.sql.Connection;
31 import java.sql.ResultSet;
32 import java.sql.SQLException;
33 import java.sql.Statement;
34
35 public class ValidateUser {
36
37     public void doPrivilegedAction(String Uname, char[] pwd)
38         throws SQLException {
39         Connection conn = getConnection();
40         if (conn == null) { // handle error
41         }
42         try {
43             String passud = hashPassword(pwd);
44             String sqlString = "SELECT * FROM db_user WHERE username = '" +
45                 Uname + "' AND password = '" + passud + "'";
46             Statement statmt = conn.createStatement();
47             ResultSet rset = statmt.executeQuery(sqlString);
48             if (!rset.next()) {
49                 throw new SQLException("Incorrect User name or password");
50             }
51         } catch (Exception e) {
52             // handle exception
53         }
54     }
55 }
```

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

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



- 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

Secure Code for SQL Injection

- In the following code, **doPrivilegedAction()** method uses **PreparedStatement** instead of **java.sql.Statement**
- 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
7 public class ValidateUser {
8
9     public void callme(String Uname, char[] pwd) throws SQLException {
10
11         Connection conn = getConnection();
12         if (conn == null) {
13         }
14         try {
15             String passud = hashPassword(pwd);
16             if (Uname.length() > 8) {
17                 }
18             String sqlString = "select * from db_user where Uname=? and pwd=?";
19             PreparedStatement prdstmt = conn.prepareStatement(sqlString);
20             prdstmt.setString(1, Uname);
21             prdstmt.setString(2, passud);
22             ResultSet rset = prdstmt.executeQuery();
23             if (!rset.next()) {
24                 throw new SQLException("Incorrect username or password");
25             }
26         }
27     }
28 }
```

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

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



Vulnerable Code for XML Injection

```
ValidateUser.java 23
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     IOException {
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

```
2 import java.util.regex.Pattern;
3
4 public class ValidateUser {
5
6     private void createXMLStream(BufferedOutputStream bufOpStrm, String quantity) throws
7     IOException {
8         if(!Pattern.matches("[0-9]+", quantity)) {
9             String xmlStr = "<item>\n<description>Widget</description>\n" + " "
10            + "<price>500.00</price>\n" + "<quantity>" + quantity + "</quantity></item>";
11            bufOpStrm.write(xmlStr.getBytes());
12            bufOpStrm.flush();
13        }
14    }
15}
```

- 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 **iNormalize** method is 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.

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 = "\u0FE64" + "script" + "\u0FE65";  
18 Pattern ptn = Pattern.compile("[<>]");  
19 Matcher mchr = ptn.matcher(Str);  
20 if (mchr.find()) {  
21     throw new IllegalStateException();  
22 } else {  
23     Str = Normalizer.normalize(Str, Form.NFKC);  
24 }  
25  
26
```

Secure Code

```
ValidateUser.java ::  
16  
17 String Str = "\u0FE64" + "script" + "\u0FE65";  
18 Str = Normalizer.normalize(Str, Form.NFKC);  
19 Pattern ptn = Pattern.compile("[<>]");  
20 Matcher mchr = ptn.matcher(Str);  
21 if (mchr.find()) {  
22     throw new IllegalStateException();  
23 } else {  
24 }
```

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     logger.severe("User login succeeded for: " + username);  
25 } else {  
26     logger.severe("User login failed for: " + username);  
27 }
```

Secure Code

```
java ::  
23 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);  
}
```

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

- In the code, validation is performed before **logging username**

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

The screenshot shows two side-by-side code editors. The left editor, labeled 'Vulnerable Code', contains the following Java code:

```
1 package Validation;
2
3 import java.util.Calendar;
4 import java.util.GregorianCalendar;
5
6 public class ValidateUser {
7     static Calendar cal=new GregorianCalendar(1995, GregorianCalendar.MAY, 23);
8
9     public static void main(String[] args) {
10
11         // args[0] is the credit card expiration date
12         // args[0] can contain either 3105, 3106 or 3107 as malicious arguments
13         // First argument prints 05 (May), second prints 23 (day) and third prints
14         // 1995 (year). If args[0] contains 3105 it prints the following line
15
16         System.out.printf(args[0] +
17             " did not match! %M%T: It was issued on %X$terd of some month", cal);
18     }
19 }
```

The line `System.out.printf(args[0] + " did not match! %M%T: It was issued on %X$terd of some month", cal);` is highlighted with a red box. The right editor, labeled 'Secure Code', contains the same code but with a different printf format:

```
1 package Validation;
2
3 import java.util.Calendar;
4 import java.util.GregorianCalendar;
5
6 public static void main(String[] args) {
7
8     // args[0] is the credit card expiration date
9     // Perform conversion with code of 16 decimal digits, i.e., the following line
10    System.out.printf("Xs did not match! %s - %M%T: It was issued on %X$terd of some month", args[0], cal);
11 }
12 }
```

The line `System.out.printf("Xs did not match! %s - %M%T: It was issued on %X$terd of some month", args[0], cal);` is highlighted with a red box. Below the code editors, there are three bullet points:

- The code describes an **information leak** issue
- It accepts the expiration date of a credit card and incorporates it within the format string
- 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

The screenshot shows a code editor with the following Java code:

```
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 readBytes(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
17        while ((bytesRead = ins.read(data, offset, data.length - offset)) != -1) {
18            offset += bytesRead;
19            str += new String(data, offset, data.length - offset, "UTF-8");
20            if (offset >= data.length) {
21                throw new IOException("Too much input");
22            }
23        }
24        ins.close();
25        return str;
26    }
27 }
```

The loop body `while ((bytesRead = ins.read(data, offset, data.length - offset)) != -1) {` is highlighted with a red box. Below the code, there is one bullet point:

- 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.

Input Validation Errors: Inappropriate Use of Split Characters in Data Structures (Cont'd)



Secure Code

- 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

```
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 readBytes(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 ::  
1 package Validation;  
2  
3 import java.io.BufferedReader;  
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 buffyStream, String quantity) throws IOException {  
13        String str = "<scr>" + "\u03ff" + "</pt>";  
14        str = Normalizer.normalize(str, Form.NFC);  
15        Pattern ptn = Pattern.compile("<script>");  
16        Matcher mchr = ptn.matcher(str);  
17        if (mchr.find()) {  
18            System.out.println("Found black listed tag");  
19        } else {  
20            str = str.replaceAll("<\\w{ASCII}\\>","");
21        }
22    }
23 }
```

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

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 21
1 import java.io.IOException;
2 import java.text.Normalizer;
3 import java.text.Normalizer.Form;
4 import java.util.regex.Matcher;
5 import java.util.regex.Pattern;
6
7 public class ValidateUser {
8
9     private void createXMLStream(BufferedOutputStream bufOpStrm, String quantity)
10        throws IOException {
11
12         String str = "<scr" + "\u0EFF" + ">pt>";
13         str = str.replaceAll("\\p{ASCII}", "\u0FFF");
14         str = Normalizer.normalize(str, Form.NFC);
15         Pattern ptn = Pattern.compile("<script>");
16         Matcher mchr = ptn.matcher(str);
17         if (mchr.find()) {
18             System.out.println("Found black listed tag");
19         } else {
20
21
22     }
23
24     str = str.replaceAll("\\p{ASCII}", "");
25 }
```

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 21
1 package Validations;
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
10        String xmlStr;
11        xmlStr = "<item>\n<description>Widget</description>\n" +
12        "<price>500.0</price>\n" +
13        "<quantity>" + quantity + "</quantity></item>";
14        bufOpStrm.write(xmlStr.getBytes());
15        bufOpStrm.flush();
16    }
17 }
```

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

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 bufOStrm, String quantity) throws IOException {
10         if(!Pattern.matches("[0-9]+", quantity)) {
11             String xmlStr = "<item>\n<description>Widget</description>\n<price>500</price>\n<quantity>" + quantity + "</quantity></item>";
12             bufOStrm.write(xmlStr.getBytes());
13             bufOStrm.flush();
14         }
15     }
16
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
3 import java.util.Date;
4 import java.text.SimpleDateFormat;
5
6 public class ValidateUser {
7
8     public static void isJune(Date date) {
9         String myString = DateFormat.getDateInstance().format(date);
10        System.out.println("The date is " + myString);
11        if(myString.startsWith("Jun ")) {
12            System.out.println("Its Cool season!");
13        } else {
14            System.out.println("Its not Cool Season");
15        }
16    }
17 }
18 
```

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

- The code snippet implements US Local while defining the date parameters

- 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

```
ValidateUser.java
1 package Validation;
2
3 import java.text.SimpleDateFormat;
4
5 public class ValidateUser {
6
7     public static void isJune(Date date) {
8         Date rightNow = null;
9         @SuppressWarnings("null")
10        String myString = DateFormat.getDateInstance(DateFormat.MEDIUM, Locale.US).format(rightNow.getTime());
11        System.out.println("The date is " + myString);
12        if(myString.startsWith("Jun ")) {
13            System.out.println("Its Cool season!");
14        } else {
15            System.out.println("Its not Cool Season");
16        }
17    }
18 }
19 
```

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

Module 04 Page 128

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

The slide features a large watermark reading 'DONOTCOPY' diagonally across the center. At the top right is the CASE logo with 'JAVA' above it. The title 'Common Secure Coding Practices' is centered below the logo. A copyright notice at the bottom right states 'Copyright © by EC-Council All Rights Reserved. Reproduction is Strictly Prohibited.'

The slide features a large watermark reading 'badalshiva@gmail.com' diagonally across the center. At the top right is the CASE logo with 'JAVA' above it. The title 'SQL Injection' is centered above a list of bullet points. Below the list are two code editor windows side-by-side, each with a red box highlighting a portion of the code. A legend at the bottom left identifies the code types: 'Insecure Code' and 'Secure Code'. A copyright notice at the bottom right states '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

```
29 PreparedStatement ps = con.prepareStatement("select * from exemplinfo where uname=? and uname=? and pwd=?");  
30 ResultSet rs = ps.executeQuery();  
31 boolean status = false;  
32 rs.next();  
33
```

Secure Code

```
23 PreparedStatement ps=con.prepareStatement("select * from exemplinfo where uname=? and pwd=?");  
24 ps.setString(1,uname);  
25 ps.setString(2,pwd);  
26 ResultSet rs=ps.executeQuery();  
27 boolean status = false;  
28 rs.next();  
29
```

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

```
File Edit Source Refactor Navigate Search Project Test Run Window Help
LoginController.jsp
22
23     PreparedStatement ps = con
24     PreparedStatement ps=con.prepareStatement("select * from ecominfo where uname=? and pwd=?");
25     ps.setString(1,uname);
26     ps.setString(2,pwd);
27     ResultSet rs=ps.executeQuery();
28
29
```

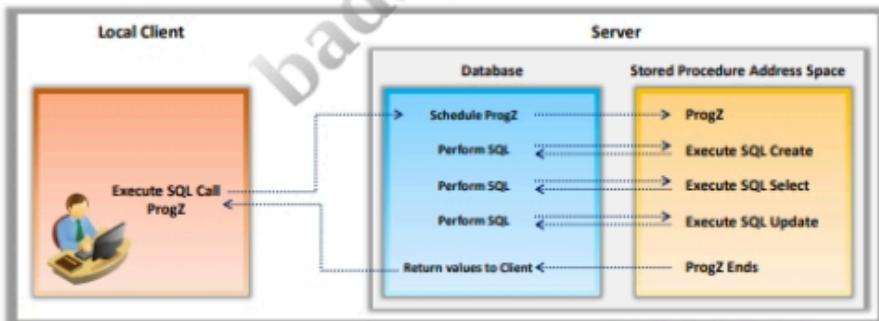
The ? is the parameter marker that represents the parameters (in JDBC)

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.

Vulnerable and Secure Code for Stored Procedures

Example: Implementing Stored Procedure

The screenshot shows two windows side-by-side. On the left is a Java IDE window for 'UserLogin.java' containing Java code that uses a CallableStatement to call a stored procedure. On the right is a 'Query Builder' window showing the SQL code for the stored procedure 'SP_AUTHENTICATE'. The code creates the procedure with parameters 'Username' and 'Passwd', and returns a result set. A note below the code states: 'The SELECT statement restricts the size of parameters and length of input variables'.

Note: `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.

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.

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

```
String query = "select * from emp where eid=?";  
PreparedStatement pst = conn.prepareStatement(query);  
  
pst.setString(2, eid);  
ResultSet rslist = pst.executeQuery();  
if (rslist != null) {  
    rslist.next();  
    ename = rslist.getString("ename");  
}
```

- The insecure code snippet queries a database for an employee id **without proper input validation mechanism** and prints the employee name

Secure Code

```
String query = "select * from emp where eid=?";  
PreparedStatement pst = conn.prepareStatement(query);  
  
pst.setString(2, eid);  
ResultSet rslist = pst.executeQuery();  
if (rslist != null) {  
    rslist.next();  
    ename = rslist.getString("ename");  
  
    if (!((String) matcher(ename)).matches(ename)) {  
        throw new ValidationException ("invalid emp name");  
    }  
}  
  
return ename;
```

- 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

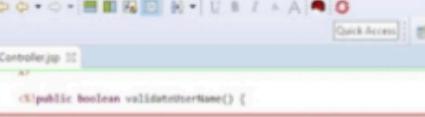
- Testing input values against a set of negative expected inputs is called **blacklisting**
- Checking all the input values received with that of the known **bad** or **negative conditions** is called blacklisting
- Developers cannot think of every possible bad input **permutation** and insert it in 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.

Vulnerable and Secure Code for Blacklisting & Whitelisting



Vulnerable Code



```
public boolean validateUserName() {
    String[] userInputs =
    new String[] { "<SCRIPT>", "SELECT", "UNION", "WHERE", "</SCRIPT>", ";", "=" };

    for (String item : userInputs) {
        if (uname.contains(item)) {
            return false;
        }
    }
    return true;
}
```

- In the code, the `validateUserName` function uses the blacklisting approach

Secure Code



The screenshot shows the Eclipse IDE interface with the Java code for `LoginController.jsp` highlighted. The code defines a method `validateUserName()` that uses a regular expression to validate a user name.

```
public boolean validateUserName() {  
    Pattern p = Pattern.compile("[a-zA-Z0-9_\\.-]*$");  
    Matcher m = p.matcher(uname);  
    boolean result = m.matches();  
    return result;  
}
```

- In the code, the `validateUserName` function uses the whitelisting approach

Copyright © by Holt, Rinehart and Winston, Inc. Additions and revisions are copyright © by The McGraw-Hill Companies, Inc.

Vulnerable and Secure Code for Blacklisting & Whitelisting (Cont'd)



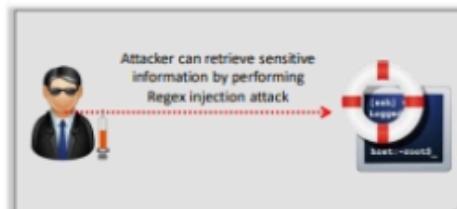
Vulnerable Code

Copyright © by IGI-Global. All Rights Reserved. Reproduction is strictly prohibited.

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
(0 1)	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 CA CO) (0-9) (0-9)	Matches AZxx, CAxx and COxx where x is a single digit

Copyright © by Holt, Rinehart and Winston, Inc. Additions and Substitutions are Reserved.

Vulnerable and Secure Code for Regular Expressions



Vulnerable Code

```
    UserAdministrator.java           conf/cassandra.yaml      application-Commands          UserLog.java
  24      // Log file is mapped into the memory and reloaded
  25
  26
  27
  28  }try {
  29     FileChannel fchannel = new FileInputStream(
  30         "path").getChannel();
  31     // Getting size of the file and mapping it
  32     int file_size = (int) fchannel.size();
  33     final ByteBuffer mappedBuffer = fchannel.map(
  34         FileChannel.MapMode.READ_ONLY, 0, file_size);
  35     Charset cs = Charset.forName("ISO-8859-1");
  36     ByteBuffer wrappedBuffer = mappedBuffer.asCharBuffer();
  37     wrappedBuffer.rewind();
  38     wrappedBuffer.limit(file_size);
  39     wrappedBuffer.position(0);
  40     wrappedBuffer.get(cs);
  41     wrappedBuffer.flip();
  42     wrappedBuffer.decode(mappedBuffer); // file is read into char buffer
  43     wrappedBuffer.rewind(); // now rewritable
  44     synchronized(lock) {
  45
  46
  47
  48
  49
  50
  51
  52
  53
  54
  55
  56
  57
  58
  59
  60
  61
  62
  63
  64
  65
  66
  67
  68
  69
  70
  71
  72
  73
  74
  75
  76
  77
  78
  79
  80
  81
  82
  83
  84
  85
  86
  87
  88
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 679
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 739
 740
 741
 742
 743
 744
 745
 746
 747
 747
 748
 749
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 809
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 849
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 939
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 949
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 979
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 989
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 999
 999
 1000
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1049
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1089
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1098
 1099
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1139
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1148
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1159
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1169
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1179
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1188
 1189
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1198
 1198
 1199
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1209
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1219
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1229
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1248
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1279
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1288
 1289
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1298
 1298
 1299
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1309
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1318
 1319
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1328
 1329
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1339
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1348
 1349
 1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1359
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1369
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378
 1379
 1379
 1380
 1381
 1382
 1383
 1384
 1385
 1386
 1387
 1388
 1388
 1389
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1398
 1399
 1399
 1400
 1401
 1402
 1403
 1404
 1405
 1406
 1407
 1408
 1409
 1409
 1410
 1411
 1412
 1413
 1414
 1415
 1416
 1417
 1418
 1419
 1419
 1420
 1421
 1422
 1423
 1424
 1425
 1426
 1427
 1428
 1429
 1429
 1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1439
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1448
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457
 1458
 1459
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1469
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1478
 1479
 1479
 1480
 1481
 1482
 1483
 1484
 1485
 1486
 1487
 1488
 1488
 1489
 1489
 1490
 1491
 1492
 1493
 1494
 1495
 1496
 1497
 1498
 1498
 1499
 1499
 1500
 1501
 1502
 1503
 1504
 1505
 1506
 1507
 1508
 1509
 1509
 1510
 1511
 1512
 1513
 1514
 1515
 1516
 1517
 1518
 1519
 1519
 1520
 1521
 1522
 1523
 1524
 1525
 1526
 1527
 1528
 1529
 1529
 1530
 1531
 1532
 1533
 1534
 1535
 1536
 1537
 1538
 1539
 1539
 1540
 1541
 1542
 1543
 1544
 1545
 1546
 1547
 1548
 1548
 1549
 1550
 1551
 1552
 1553
 1554
 1555
 1556
 1557
 1558
 1559
 1559
 1560
 1561
 1562
 1563
 1564
 1565
 1566
 1567
 1568
 1569
 1569
 1570
 1571
 1572
 1573
 1574
 1575
 1576
 1577
 1578
 1579
 1579
 1580
 1581
 1582
 1583
 1584
 1585
 1586
 1587
 1588
 1588
 1589
 1589
 1590
 1591
 1592
 1593
 1594
 1595
 1596
 1597
 1598
 1598
 1599
 1599
 1600
 1601
 1602
 1603
 1604
 1605
 1606
 1607
 1608
 1609
 1609
 1610
 1611
 1612
 1613
 1614
 1615
 1616
 1617
 1618
 1619
 1619
 1620
 1621
 1622
 1623
 1624
 1625
 1626
 1627
 1628
 1629
 1629
 1630
 1631
 1632
 1633
 1634
 1635
 1636
 1637
 1638
 1639
 1639
 1640
 1641
 1642
 1643
 1644
 1645
 1646
 1647
 1648
 1648
 1649
 1650
 1651
 1652
 1653
 1654
 1655
 1656
 1657
 1658
 1659
 1659
 1660
 1661
 1662
 1663
 1664
 1665
 1666
 1667
 1668
 1669
 1669
 1670
 1671
 1672
 1673
 1674
 1675
 1676
 1677
 1678
 1679
 1679
 1680
 1681
 1682
 1683
 1684
 1685
 1686
 1687
 1688
 1688
 1689
 1689
 1690
 1691
 1692
 1693
 1694
 1695
 1696
 1697
 1698
 1698
 1699
 1699
 1700
 1701
 1702
 1703
 1704
 1705
 1706
 1707
 1708
 1709
 1709
 1710
 1711
 1712
 1713
 1714
 1715
 1716
 1717
 1718
 1719
 1719
 1720
 1721
 1722
 1723
 1724
 1725
 1726
 1727
 1728
 1729
 1729
 1730
 1731
 1732
 1733
 1734
 1735
 1736
 1737
 1738
 1739
 1739
 1740
 1741
 1742
 1743
 1744
 1745
 1746
 1747
 1748
 1748
 1749
 1750
 1751
 1752
 1753
 1754
 1755
 1756
 1757
 1758
 1759
 1759
 1760
 1761
 1762
 1763
 1764
 1765
 1766
 1767
 1768
 1769
 1769
 1770
 1771
 1772
 1773
 1774
 1775
 1776
 1777
 1778
 1779
 1779
 1780
 1781
 1782
 1783
 1784
 1785
 1786
 1787
 1788
 1788
 1789
 1789
 1790
 1791
 1792
 1793
 1794
 1795
 1796
 1797
 1798
 1798
 1799
 1799
 1800
 1801
 1802
 1803
 1804
 1805
 1806
 1807
 1808
 1809
 1809
 1810
 1811
 1812
 1813
 1814
 1815
 1816
 1817
 1818
 1819
 1819
 1820
 1821
 1822
 1823
 1824
 1825
 1826
 1827
 1828
 1829
 1829
 1830
 1831
 1832
 1833
 1834
 1835
 1836
 1837
 1838
 1839
 1839
 1840
 1841
 1842
 1843
 1844
 1845
 1846
 1847
 1848
 1848
 1849
 1850
 1851
 1852
 1853
 1854
 1855
 1856
 1857
 1858
 1859
 1859
 1860
 1861
 1862
 1863
 1864
 1865
 1866
 1867
 1868
 1869
 1869
 1870
 1871
 1872
 1873
 1874
 1875
 1876
 1877
 1878
 1879
 1879
 1880
 1881
 1882
 1883
 1884
 1885
 1886
 1887
 1888
 1888
 1889
 1889
 1890
 1891
 1892
 1893
 1894
 1895
 1896
 1897
 1898
 1898
 1899
 1899
 1900
 1901
 1902
 1903
 1904
 1905
 1906
 1907
 1908
 1909
 1909
 1910
 1911
 1912
 1913
 1914
 1915
 1916
 1917
 1918
 1919
 1919
 1920
 1921
 1922
 1923
 1924
 1925
 1926
 1927
 1928
 1929
 1929
 1930
 1931
 1932
 1933
 1934
 1935
 1936
 1937
 1938
 1939
 1939
 1940
 1941
 1942
 1943
 1944
 1945
 1946
 1947
 1948
 1948
 1949
 1950
 1951
 1952
 1953
 1954
 1955
 1956
 1957
 1958
 1959
 1959
 1960
 1961
 1962
 1963
 1964
 1965
 1966
 1967
 1968
 1969
 1969
 1970
 1971
 1972
 1973
 1974
 1975
 1976
 1977
 1978
 1979
 1979
 1980
 1981
 1982
 1983
 1984
 1985
 1986
 1987
 1988
 1988
 1989
 1989
 1990
 1991
 1992
 1993
 1994
 1995
 1996
 1997
 1998
 1998
 1999
 1999
 2000
 2001
 2002
 2003
 2004
 2005
 2006
 2007
 2008
 2009
 2009
 2010
 2011
 2012
 2013
 2014
 2015
 2016
 2017
 2018
 2019
 2019
 2020
 2021
 2022
 2023
 2024
 2025
 2026
 2027
 2028
 2029
 2029
 2030
 2031
 2032
 2033
 2034
 2035
 2036
 2037
 2038
 2039
 2039
 2040
 2041
 2042
 2043
 2044
 2045
 2046
 2047
 2048
 2048
 2049
 2050
 2051
 2052
 2053
 2054
 2055
 2056
 2057
 2058
 2059
 2059
 2060
 
```

- 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



The screenshot shows a Java code editor with the following code:

```
// Dynamically construct regex using user string
String regexp = ".*" + public\\((\\w+\\)\\) +.*" + search + ".*)";
Pattern exprsPattern = Pattern.compile(regexp);
Matcher match = exprsPattern.matcher(chBuff);
while (match.find()) {
    String Found = match.group(1);
```

The line `String regexp = ".*" + public\\((\\w+\\)\\) +.*" + search + ".*)";` is highlighted with a red box.

- This secure code filters the `non-alphanumeric characters` excluding `'space'` and `'single quote'` from the search string. This code prevents regexp injection

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     public static void main(String args[]) {
8         FileInputStream finStr = null;
9         try {
10             finStr = new FileInputStream("Anyfile");
11             DataInputStream dinStr = new DataInputStream(finStr);
12             byte[] bData = new byte[1024];
13
14             dinStr.readFully(bData);
15             String result = new String(bData);
16         }
17     }
18 }
```

- In the code, the difference between **default encoding** and **byte array** produced during encoding causes **undefined behavior** for inputs

Secure Code

```
CharacterEncoding.java
1 Import java.io.FileInputStream;
2 Import java.io.IOException;
3
4 public class Characterencoding {
5
6     public static void main(String args[]) {
7         FileInputStream finStr = null;
8         try {
9             finStr = new FileInputStream("Anyfile");
10            DataInputStream dinStr = new DataInputStream(finStr);
11            byte[] bData = new byte[1024];
12
13            dinStr.readFully(bData);
14            String encoding="UTF-16LE";
15            String result = new String(bData,encoding);
16        }
17    }
18 }
```

- The code explicitly specifies intended **character encoding** in the **String constructor**

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

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:22
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.org/TR/REC-html40/loose.dtd">
```

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

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

```
1 package com.cse;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8
9
10 public class Result extends HttpServlet {
11     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
12         String result=request.getParameter("result");
13
14         Student Result=&nbs;<result>;
15
16     }
17 }
```

- 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

```
1 package com.cse;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8
9
10 public class HTMLEncoder extends HttpServlet {
11     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
12         final StringTagFragment res = new StringTagFragment();
13         final StringCharacterIterator itr = new StringCharacterIterator(TagFragment);
14         char chctr = itr.current();
15         while (chctr != StringCharacterIterator.DONE) {
16             if (chctr == '<')
17                 res.append("&lt;"); // encode less than sign
18             else if (chctr == '>')
19                 res.append("&gt;"); // encode greater than sign
20             else if (chctr == '"')
21                 res.append("&quot;"); // encode double quote
22             else if (chctr == '&apos;')
23                 res.append("&apos;"); // encode single quote
24             else if (chctr == '&#039;')
25                 res.append("&#039;"); // encode apostrophe
26             else if (chctr == '&#034;')
27                 res.append("&#034;"); // encode double quote
28             else if (chctr == '&#039;')
29                 res.append("&#039;"); // encode single quote
30             else if (chctr == '&#034;')
31                 res.append("&#034;"); // encode double quote
32             else if (chctr == '&#039;')
33                 res.append("&#039;"); // encode single quote
34             else if (chctr == '&#034;')
35                 res.append("&#034;"); // encode double quote
36             else if (chctr == '&#039;')
37                 res.append("&#039;"); // encode single quote
38             else if (chctr == '&#034;')
39                 res.append("&#034;"); // encode double quote
40             else { // chctr is not a special one
41                 // add it to the result as is
42                 res.append(chctr);
43             }
44             chctr = itr.next();
45         }
46         return res.toString();
47     }
48 }
```

- 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.

HTML Encoding using ESAPI Encoder

Vulnerable Code

```
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 "-//IETF//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
7    1999/12/rfc3629.html">
8 <html>
9 <head>
10 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
11 <title>Simple Captcha</title>
12 </head>
13 <body>
14
15 <%
16     String result=request.getParameter("result");
17
18     Student Result:<%= result%>
19
20 </body>
21 </html>
22
```

Secure Code

```
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 "-//IETF//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
7    1999/12/rfc3629.html">
8 <html>
9 <head>
10 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
11 <title>Simple Captcha</title>
12 </head>
13 <body>
14
15 <%
16     String result=ESAPI.encoder().encodeForHTML(request.getParameter("result"));
17
18     Student Result:<%= result%>
19
20 </body>
21 </html>
22
```

- 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**

- The given code implements **ESAPIEncoder** 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

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1"%>
3 <%@ page import="java.io.*" %>
4 <%@ page import="javax.servlet.*" %>
5 <%@ page import="javax.servlet.http.*" %>
6 <%@ page import="java.util.*" %>
7
8 protected void doGet(HttpServletRequest request, HttpServletResponse response)
9     throws ServletException, IOException {
10
11     doPost(request, response);
12 }
```

Secure Code

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1"%>
3 <%@ page import="java.io.*" %>
4 <%@ page import="javax.servlet.*" %>
5 <%@ page import="javax.servlet.http.*" %>
6 <%@ page import="java.util.*" %>
7
8 protected void doGet(HttpServletRequest request, HttpServletResponse response)
9     throws ServletException, IOException {
10
11     response.sendRedirect(ERROR_PAGE);
12 }
```

- In the code, attackers can **manipulate the GET request** to view any unauthorized data
- 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.

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 **CSRGuard 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 f=new File(System.getProperty("user.dir")+\System.getProperty("file.separator")+args[0]);
10        String absolutPath = f.getAbsolutePath();
11        if (absolutPath.startsWith(Path.of(args[0]).getAbsolutePath())) {
12            throw new IllegalArgumentException();
13        }
14        if (!isValidAbsolutePath()) {
15            // Validation Check
16            throw new IllegalArgumentException();
17        }
18    }
19 }
```

- The code snippet uses command line argument for file path and uses **File.getAbsolutePath()** method to find the absolute path

Secure 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 f=new File(System.getProperty("user.dir")+\System.getProperty("file.separator")+args[0]);
10        String canonicalPath = f.getCanonicalPath();
11        if (canonicalPath.startsWith(Path.of(args[0]).getAbsolutePath())) {
12            throw new IllegalArgumentException();
13        }
14        if (!isValidAbsolutePath()) {
15            // Validation Check
16            throw new IllegalArgumentException();
17        }
18    }
19 }
```

- The **getCanonicalPath()** method is used to resolve any aliases, symbolic links and shortcuts

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

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

Downloaded by [University of California, San Diego] at 00:00 09 September 2016

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

Secure Code

```
41  
42  
43  
44  
45     response.sendRedirect(Const.TOGGLE_SSO_URL + "returnUrl=" + Const.GOTO_FID + "6" + Const.FID_0_0);  
46  
47 }
```

- The code shows **cookie header response split**

```
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 Houghton Mifflin Company. All Rights Reserved. Reproduction is strictly prohibited.

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

```
CharacterEncoding.java 22
33 final byte CR=(byte) '\r';
34 //This represents CR byte
35 final byte LF=(byte) '\n';
36 //This represents LF byte
37 String cleanstrn4httpHeader(final String P_shttpHeaderStr){
38     if(P_shttpHeaderStr==null)
39         return P_shttpHeaderStr;
40     return fullfindReplace(fullfindReplace(P_shttpHeaderStr, "\r", ""), "\n", "");
41 }
42 /**
43 * All the CR/LF characters strings are cleaned by this function in attack
44 * HTTP Header needs to include this String @param P_shttpHeaderStr
45 * @string in CRLF is cleaned by return */
46 }
```

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

```
MyShoppingCart.java 34
34 String rptName=request.getParameter("nameofreport");
35 File rptfile = new File("/usr/local/apfr/reports/" + rptName);
36
37 //
```

- An attacker can **manipulate** the file name or **delete** desired one

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

Protecting Application from Log Injection Attack

Vulnerable Code

```
else if (loginIncorrect) {
    logger.severe("User login succeeded for: " + username);
} else {
    logger.severe("User login failed for: " + username);
```

Secure Code

```
else if (loginIncorrect) {
    logger.severe("User login succeeded for: " + sanitizeUser(username));
} else {
    logger.severe("User login failed for: " + sanitizeUser(username));
```

```
public static String sanitizeUser(String username) {
    return Pattern.matches("[A-Za-z-\\s-0-9]+", 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
```

Secure Code

- There are different ways of handling log injection attacks
 - Whitelisting
 - Blacklisting

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

XML Injection

Insecure Code

```
String xmlstr = "item</description><id>" + id + "</id><description>" + description + "</description><price>" + price + "</price>" + "<quantity>" + quantity + "</quantity></item>";
```

Secure Code

```
if(!Pattern.matches("[0-9]", quantity)) {
```

Insecure Code

- This code uses string concatenation to build an XML query, but fails to validate the input resulting in an XML attack

Secure Code

- 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.

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

```
path.java 10
17 String bktype = request.getParameter("bktype");
18 String cmd = new String("cmd|exe /K\c:\util1\my0h.bat"+ bktype + "BkC:\Util1\cleanup.bat\"");
19 Runtime.getRuntime().exec(cmd);
```

- In the code, attackers can execute **multiple commands** separated by two ampersands passing a string "**cmd|exe c:\\dbms*.***"

Secure Code

```
UserLogin.java 126
326
327     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
328         final int MaxName = 50;
329         final String f_regexp = "[a-zA-Z]{1,"+MaxName+"}";
330         final Pattern bpattern = Pattern.compile(f_regexp);
331         final String Username = request.getParameter("username");
332         validateName(Username, bpattern);
333
334         String bktype = request.getParameter("bktype");
335         String cmd = new String("cmd|exe /K\c:\util1\my0h.bat"+ bktype + "BkC:\Util1\cleanup.bat");
336         Runtime.getRuntime().exec(cmd);
337         doGet(request, response);
338     }
```

- 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

```
path.java 10
1 ~ imports javax.naming.directory.Attributes;
2 import javax.naming.NamingException;
3 import javax.naming.directory.Attribute;
4 import javax.naming.directory.Attributes;
5 import javax.naming.directory.DirContext;
6 import javax.naming.directory.InitialDirContext;
7 import javax.naming.directory.SearchControls;
8 import javax.naming.directory.SearchResult;
9 import javax.naming.directory.SearchResult;
10 import javax.naming.directory.SearchControls;
11 import javax.naming.directory.SearchResult;
12 import javax.xml.xpath.XPathExpression;
13
14 public class path { // String UserName = "S"; // Invalid
15 // String Upwd = "+" // Invalid
16
17     private void searchRecord(String UserName, String Upwd) throws NamingException {
18         Hashtable<String, String> environment = new Hashtable<String, String>();
19         environment.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFact
try {
20             DirContext dircx = new InitialDirContext(environment);
21             SearchControls searchctrl = new SearchControls();
22             String[] attrFilter = {"cn", "email"};
23             searchctrl.setReturningAttributes(attrFilter);
24             searchctrl.setSearchScope(SearchControls.SUBTREE_SCOPE);
25             String bsc = "dc=con-example, dc=com";
26             String fltr = "(&(cn=" + UserName + ")(&(upn=" + Upwd + "+))";
27             // This results to (&(cn=S)(upn=+))
28             NamingEnumeration< SearchResult> results = dircx.search(bsc, fltr, searchctrl);
29             while (results.hasMore()) {
30                 SearchResult schr = (SearchResult) results.next();
31                 Attributes attrs = schr.getAttributes();
32                 Attribute attr = attrs.get("cn");
33                 System.out.println(attr.get());
34                 attr = attrs.get("email");
35                 System.out.println(attr.get());
36             }
37             dircx.close();
38         } catch(Exception e) { e
39     }
40 }
```

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

LDAP Injection (Cont'd)

The diagram illustrates two scenarios involving an LDAP Server. In the first scenario, a 'Client' (represented by a person icon) sends a 'Normal Query + Code Injection' (indicated by a red X) to the 'LDAP Server'. The server returns 'Normal Result and/or Additional Information'. In the second scenario, a 'Client' sends a 'Normal Query' (indicated by a green checkmark) to the 'LDAP Server', which returns a 'Normal Result'.

Secure Code:

Sanitize Inputs using Strong String Filter

```
path.java :: 9
10 public static XPathExpression compile(String string) {
11
12     String UsrName = "Sherlock Holmes"; // Valid
13     String Usrpwd = "secret2"; // Valid
14
15     srchctr1.setSearchScope(SearchControls.SUBTREE_SCOPE);
16     String bsc = "dircon@example, dc=example, dc=com";
17     if(!UsrName.matches("[\\w\\\\s]*") || !Usrpwd.matches("[\\w]*")) {
18         throw new IllegalArgumentException("Invalid input");
19     }
20
21     String fltr = "(" + strn + "=" + UsrName + ")(&userPassword=" + Usrpwd + ")";
22 }
```

The code uses **whitelisting technique** to sanitize inputs using strong String filter that contains valid characters

XML External Entity Attack

An **XML External Entity attack** enables the attacker to access local or remote contents resulting in information disclosure and denial of service attacks.

The external entities use SYSTEM Keyword; the XML parser reads the document from the URI specified.

This value can be replaced by the attacker to read any local or remote file.

The diagram shows a flow from an 'Attacker' (hacker icon) to a 'Client' (web browser). The 'Client' has a 'Login' form. An 'XXE' (grey circle) is placed between the Attacker and the Client. The Client sends a request to an 'XML Parser' (red square). Another 'XXE' is placed between the Client and the XML Parser. The XML Parser returns a response to the Client. A third 'XXE' is placed between the XML Parser and the Attacker. The Attacker sends a file path ('/etc/passwd') to the XML Parser via the third XXE.

XML External Entity Attack (Cont'd)



- Prevent XML External Entity attacks by implementing `org.xml.sax.EntityResolver` `resolveEntity()` method
- Contains customized handler using whitelist for checking external entities
- `resolveEntity()` method resolves the external entities of XML document while parsing

Insecure Code

```
FileValidation.java  XValidation.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 XValidation {
15
16     private static DefaultHandler DefaultHandler;
17
18     public static void receiveInputStream(InputStream instance, DefaultHandler defaultHandler)
19             throws ParserException, SAXException, IOException, org.xml.sax.SAXException {
20
21         SAXParserFactory factory = SAXParserFactory.newInstance();
22         try {
23             SAXParser saxParser = factory.newSAXParser();
24         } catch (ParserConfigurationException e) {
25             e.printStackTrace();
26         } catch (IOException e) {
27             e.printStackTrace();
28         } catch (org.xml.sax.SAXException e) {
29             // TODO Auto-generated catch block
30             e.printStackTrace();
31         }
32         saxParser.parse(instance, defaultHandler);
33     }
34
35     public static void main(String args[]) throws ParserException, SAXException, IOException {
36         try {
37             receiveInputStream(new FileInputStream("evil.xml"), DefaultHandler);
38         } catch (Exception e) {
39             // TODO handle exception
40         }
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  XValidation.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        String entityPath="file:/Users/onlinestore/good.xml";
14        if(systemId.equals(entityPath)) {
15            System.out.println("Resolving entity: "+publicId +" "+ systemId);
16            return new InputSource(entityPath);
17        } else {
18            return new InputSource();
19        }
20    }
21
22 }
```

resolveEntity() method Syntax

```
InputSource resolveEntity(String publicId, String systemId) throws SAXException, IOException
```

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

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 22
1 import java.util.regex.Matcher;
2 import java.util.regex.Pattern;
3
4 public class FileValidation {
5
6     private static Pattern fileExtnPtn = Pattern.compile("(^\\s*(\\..{2})\\.(txt|doc|csv|pdf)$)");
7
8     public static boolean validatefileExtn(String userName){
9
10         Matcher mch = fileExtnPtn.matcher(userName);
11         if(mch.matches()){
12             return true;
13         }
14         return false;
15     }
16
17     public static void main(String args){
18         System.out.println("Is 'JavaNovice.pdf' allowed file? "
19                             +validatefileExtn("JavaNovice.pdf"));
20         System.out.println("Is 'eric.doc' allowed file? "
21                             +validatefileExtn("eric.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.

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)

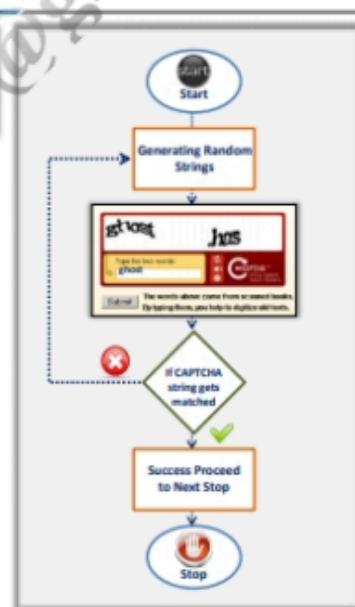
Used to **authenticate** user input

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.

Sample Code for Creating CAPTCHA



The creation of CAPTCHA is given in the Servlet code `CaptchaServlet.java`



```
1 Login.jsp 2 index.jsp 3 result.jsp 4 CaptchaServlet... 5 CaptchaServ... 6
7 import java.io.OutputStream;
8 import java.util.Random;
9
10 import javax.imageio.ImageIO;
11 import javax.servlet.ServletException;
12 import javax.servlet.http.HttpServlet;
13 import javax.servlet.http.HttpServletRequest;
14 import javax.servlet.http.HttpServletResponse;
15 import java.awt.Color;
16 import java.awt.Font;
17 import java.awt.GradientPaint;
18 import java.awt.Graphics2D;
19 import java.awt.RenderingHints;
20 import java.awt.image.BufferedImage;
21
22 /**
23  * Servlet implementation class CaptchaServlet
24 */
25 public class CaptchaServlet extends HttpServlet {
26     private static final long serialVersionUID = 3L;
27
28     protected void doGet(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] = getARandomNumber(8).toCharArray();
34         final BufferedImage buffImage = new BufferedImage(wd, ht,
35                 BufferedImage.TYPE_3BYTE_BGR);
36         final Graphics2D G2D = buffImage.createGraphics();
37         G2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
38                 RenderingHints.VALUE_ANTIALIAS_ON);
39         G2D.setRenderingHint(RenderingHints.KEY_RENDERING,
40                 RenderingHints.VALUE_RENDER_QUALITY);
41         G2D.setRenderingHint(RenderingHints.VALUE_RENDER_QUALITY);
42         G2D.setFont(F1);
43         G2D.setColor(new Color(255, 255, 0));
44
45     }
46
47     protected void execute(HttpServletRequest request,
48             HttpServletResponse response) throws ServletException, IOException {
49         final int wd = 200;
50         final int ht = 50;
51         char cdata[][] = new char[1][];
52         cdata[0] = getARandomNumber(8).toCharArray();
53         Random rand = new Random();
54         final String captcha = String.valueOf(cdata[0]);
55         request.getSession().setAttribute("captcha", captcha);
56         int xCoordinate = 0;
57         int yCoordinate = 0;
58         for (int a = 0; a < cdata[0].length(); a++) {
59             xCoordinate += 10 + (Math.abs(rand.nextInt()) % 15);
60             if (xCoordinate > wd - 5) {
61                 xCoordinate = 0;
62             }
63             yCoordinate += 20 * Math.abs(rand.nextInt()) % 20;
64             G2D.drawString(cdata[0], a, xCoordinate, yCoordinate);
65         }
66         G2D.dispose();
67         response.setContentType("image/png");
68
69         final OutputStream outStr = response.getOutputStream();
70         ImageIO.write(buffImage, "png", outStr);
71         outStr.close();
72     }
73
74     protected void doPost(HttpServletRequest request,
75             HttpServletResponse response) throws ServletException, IOException {
76         execute(request, response);
77     }
78
79     protected void doGet(HttpServletRequest request,
80             HttpServletResponse response) throws ServletException, IOException {
81         final int length = 8;
82         Random rand = new Random();
83         char[] buf = new char[length];
84         for (int i = 0; i < length; i++) {
85             buf[i] = chode.charAt(rand.nextInt(chode.length()));
86         }
87         return new String(buf);
88     }
89 }
```

Source: <http://techbycaltech.com>

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

Sample Code for Creating CAPTCHA (Cont'd)



```
1 Login.jsp 2 index.jsp 3 result.jsp 4 CaptchaServlet... 5 CaptchaServ... 6
7
8 final GradientPaint gradPoint = new GradientPaint(0, 0, ht / 2, 0,
9         Color.black, true);
10 G2D.setPaint(gradPoint);
11 G2D.fillRect(0, 0, wd, ht);
12 G2D.setColor(new Color(255, 153, 0));
13 Random rand = new Random();
14 final String captcha = String.valueOf(cdata[0]);
15 request.getSession().setAttribute("captcha", captcha);
16 int xCoordinate = 0;
17 int yCoordinate = 0;
18 for (int a = 0; a < cdata[0].length(); a++) {
19     xCoordinate += 10 + (Math.abs(rand.nextInt()) % 15);
20     if (xCoordinate > wd - 5) {
21         xCoordinate = 0;
22     }
23     yCoordinate += 20 * Math.abs(rand.nextInt()) % 20;
24     G2D.drawString(cdata[0], a, xCoordinate, yCoordinate);
25 }
26 G2D.dispose();
27 response.setContentType("image/png");
28
29 final OutputStream outStr = response.getOutputStream();
30 ImageIO.write(buffImage, "png", outStr);
31 outStr.close();
32 }
33
34 protected void doGet(HttpServletRequest request,
35             HttpServletResponse response) throws ServletException, IOException {
36     execute(request, response);
37 }
38
39 protected void doPost(HttpServletRequest request,
40             HttpServletResponse response) throws ServletException, IOException {
41     execute(request, response);
42 }
43
44 protected void doGet(HttpServletRequest request,
45             HttpServletResponse response) throws ServletException, IOException {
46     execute(request, response);
47 }
48
49 protected void doPost(HttpServletRequest request,
50             HttpServletResponse response) throws ServletException, IOException {
51     execute(request, response);
52 }
53
54 protected void doGet(HttpServletRequest request,
55             HttpServletResponse response) throws ServletException, IOException {
56     execute(request, response);
57 }
58
59 protected void doPost(HttpServletRequest request,
60             HttpServletResponse response) throws ServletException, IOException {
61     execute(request, response);
62 }
63
64 protected void doGet(HttpServletRequest request,
65             HttpServletResponse response) throws ServletException, IOException {
66     execute(request, response);
67 }
68
69 protected void doPost(HttpServletRequest request,
70             HttpServletResponse response) throws ServletException, IOException {
71     execute(request, response);
72 }
73
74 protected void doGet(HttpServletRequest request,
75             HttpServletResponse response) throws ServletException, IOException {
76     execute(request, response);
77 }
78
79 protected void doPost(HttpServletRequest request,
80             HttpServletResponse response) throws ServletException, IOException {
81     execute(request, response);
82 }
83
84 protected void doGet(HttpServletRequest request,
85             HttpServletResponse response) throws ServletException, IOException {
86     execute(request, response);
87 }
88
89 protected void doPost(HttpServletRequest request,
90             HttpServletResponse response) throws ServletException, IOException {
91     execute(request, response);
92 }
93
94 protected void doGet(HttpServletRequest request,
95             HttpServletResponse response) throws ServletException, IOException {
96     execute(request, response);
97 }
98
99 protected void doPost(HttpServletRequest request,
100            HttpServletResponse response) throws ServletException, IOException {
101    execute(request, response);
102 }
```

Source: <http://techbycaltech.com>

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

Sample Code for CAPTCHA Verification



```
30 import java.io.IOException;
31 import javax.servlet.ServletException;
32 import javax.servlet.annotation.WebServlet;
33 import javax.servlet.http.HttpServlet;
34 import javax.servlet.http.HttpServletRequest;
35 import javax.servlet.http.HttpServletResponse;
36 import javax.servlet.http.HttpSession;
37 @WebServlet("/Captcha")
38 public class CaptchaVerification extends HttpServlet {
39     private static final long serialVersionUID = 1L;
40
41
42     public CaptchaVerification() {
43         super();
44     }
45
46
47     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
48
49         final String vCaptcha = (String) request.getSession().getAttribute("captcha");
50         final String code = (String) request.getParameter("code");
51         if (vCaptcha != null && code != null) {
52             if (vCaptcha.equals(code)) {
53                 request.setAttribute("result", "Congratulations, You Passed The Captcha Test");
54             } else {
55                 request.setAttribute("result",
56                     "Sorry, You Failed The Captcha Test");
57             }
58             request.getRequestDispatcher("/result.jsp").forward(request,
59             response);
60         }
61     }
62 }
```

Copyright © by Holt McDougal. All rights reserved. Reproduction is strictly prohibited.

Sample Code for Displaying CAPTCHA



The CAPTCHA is displayed using 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 <form method="post" action="verifyCaptcha.do">
9 <table>
10 <tr>
11 <td align="center">
12 <td align="center">
13 
14 <br />
15 <br />
16 <input type="text" name="captchacode">
17 </td>
18 </tr>
19 </table>
20 <input type="submit" value="submit">
21 </form>
22 <br>
23 <br>
24
25 </body>
26 </html>
27
```

```
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.org/TR/html4/loose.dtd">
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 <cl>request.getAttribute("result")</cl>
11 </body>
12 </html>
```

Comments to the **Bill Tracker** will be reviewed. Several actions to improve the update.

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.

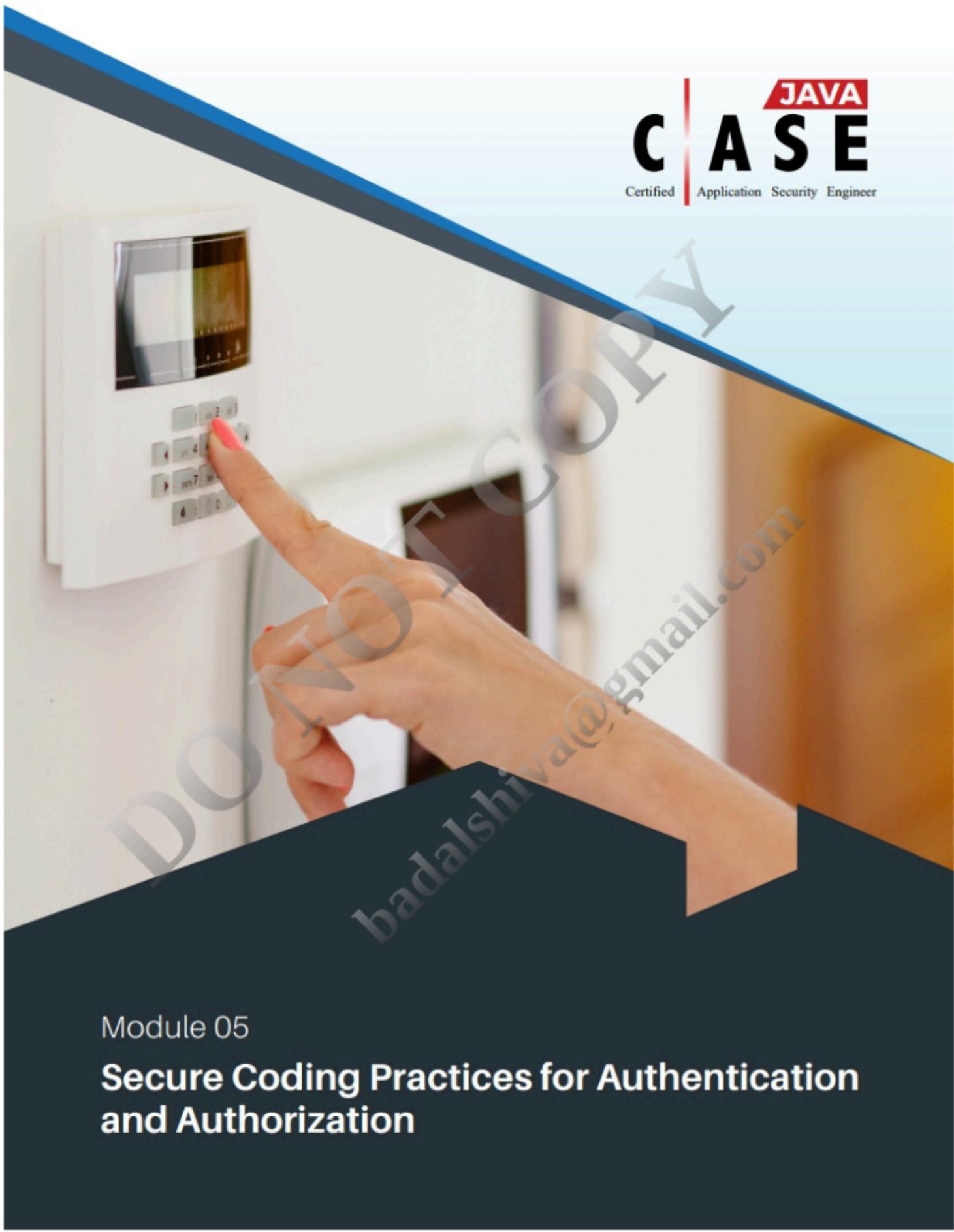
Module Summary



- 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.

DO NOT COPY
This page is intentionally left blank.
badalshiva@gmail.com



Module 05

Secure Coding Practices for Authentication and Authorization

DO NOT COPY
This page is intentionally left blank.
badalshiva@gmail.com

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.

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

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**

Authentication Mechanism Implementation



```
File Edit Source Refactor Navigate Search Project Run Window Help
User.java
public void SignIn( ) {
    if (account.authenticate(username, password) == null) {
        message = "Invalid user Username or password!";
    } else {
        message = "SignIn successful";
        FacesContext fcontext = FacesContext.getCurrentInstance();
        HttpServletRequest req = (HttpServletRequest) fcontext.getExternalContext().getRequest();

        try {
            req.login(this.message, this.password);
            Principal p = FacesContext.getCurrentInstance().getExternalContext().getUserPrincipal();
            Username = p.getName();
        } catch(ServletException ex) {
            fcontext.addMessage(null, new FacesMessage("Unknown Login"));
        }
    }
}

```

Types of Authentication

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**

```
<web-app>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>PrivateAndSensitive</web-resource-name>
      <url-pattern>/private/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>executive</role-name>
      <role-name>admin</role-name>
    </auth-constraint>
    <!-- Form based authorization -->
    <login-config>
      <auth-method>FORM</auth-method>
      <form-login-config>
        <form-login-page>/login.jsp</form-login-page>
        <form-error-page>/error.jsp</form-error-page>
      </form-login-config>
    </login-config>
  </security-constraint>
</web-app>
```

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

Programmatic Security Implementation



```
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 @WebServlet("/LoginServlet")
14 public class LoginServlet extends HttpServlet {
15     private static final long serialVersionUID = 3L;
16     public LoginServlet() {
17         super();
18     }
19
20     protected void processRequest(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
21     {
22
23         resp.setContentType("text/html;charset=UTF-8");
24         PrintWriter pw = resp.getWriter();
25         try {
26             String UsrName = req.getParameter("txtUserName");
27             String passd = req.getParameter("txtPassword");
28             pw.println("Before Login..." + "<br><br>");
29             pw.println("IsUserInRole..." + "<br>" + req.isUserInRole("ECCUser") + "<br>");
30             pw.println("getRemoteUser?..." + req.getRemoteUser() + "<br>");
31             pw.println("getUserPrincipal?..." + "<br>" + req.getUserPrincipal() + "<br>");
32             pw.println("getAuthType?..." + req.getAuthType() + "<br><br>");
33         } catch (Exception e) {
34
35     }
36
37     }
38
39     }
40
41
42     try {
43         req.login(UsrName, passd);
44     } catch(ServletException se) {
45         pw.println("Login Failed with a ServletException.."
46         + se.getMessage());
47         return;
48     }
49
50     pw.println("After Login..." + "<br><br>");
51     pw.println("IsUserInRole?..." + "<br>" + req.isUserInRole("ECCUser") + "<br>");
52     pw.println("getRemoteUser?..." + req.getRemoteUser() + "<br>");
53     pw.println("getUserPrincipal?..." + "<br>" + req.getUserPrincipal() + "<br>");
54     pw.println("getAuthType?..." + req.getAuthType() + "<br><br>");
```

Programmatic Security Implementation (Cont'd)



```
55
56
57     }
58
59     }
60
61     }
62
63     }
64
65     }
66
67     }
68
69     }
70
71     }
72
73     }
74
75     }
76
77     }
78
79     }
80
81     }
82
83     }
84
85     }
86
87     }
88
89     }
90
91     }
92
93     }
94
95     }
96
97     }
98
99     }
100 }
```

Java EE Authentication Implementation Example



```
File Edit Refactor Source Navigate Search Project Run Window Help
User.java
46 public void Userlogin( ) {
47     if (account.authenticate(username, password) == null) {
48         message = "Invalid user Username or password!";
49     } else {
50         message = "Login successful";
51         FacesContext fcontext = FacesContext.getCurrentInstance();
52         HttpServletRequest req = (HttpServletRequest) fcontext.getExternalContext().getRequest();
53
54         try {
55             req.login(this.message, this.password);
56             Principal p = FacesContext.getCurrentInstance().getExternalContext().getUserPrincipal();
57             Username = p.getName();
58         } catch(ServletException ex) {
59             {
60                 fContext.addMessage(null, new FacesMessage("Unknown User"));
61             }
62         }
63     }
64 }
65
66 }
```

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

HTTP Basic Authentication



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>
```

web.xml

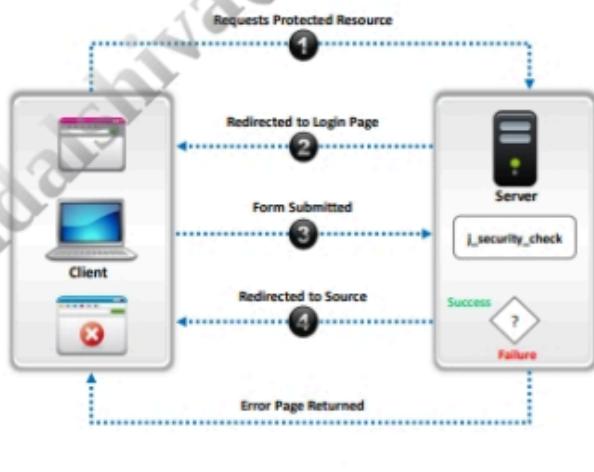
```
21 <security-constraint>
22 </security-constraint>
23<login-config>
24 <auth-method>BASIC</auth-method>
25 <realm-name>MyRealm</realm-name>
26</login-config>
27</web-app>
```

- Define roles and setup users with usernames, passwords, and roles
- Activating BASIC authentication in the server and designation of Realm
- Implementing BASIC Authentication with SSL

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 Implementation

The screenshot shows a Java Development Kit (IDE) interface. On the left, a code editor window displays the XML file `tomcat-users.xml` with the following content:

```
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>
```

On the right, another code editor window displays the XML file `web.xml` with the following content:

```
22 </security-constraint>
23<login-config>
24<auth-method>FORM</auth-method>
25<form-login-config>
26 <form-login-page>/login.jsp</form-login-page>
27 <form-error-page>/login-error.html</form-error-page>
28</form-login-config>
29</login-config>
30</web-app>
```

Below the windows, a list of steps is shown:

- 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

The diagram illustrates the Working Overview of Kerberos with three numbered steps:

- 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.

Secured Kerberos Implementation



Example for Kerberos Implementation with **jaas.conf**

```
File Edit Navigate Search Project Run Window Help
jaas.conf x
7 Server {
8     com.sun.security.auth.module.Krb5LoginModule required
9         useKeyTab=false
10        storeKey=true
11        useTicketCache=false
12        principal="webserver/bully@eccouncil.org";
13};
```

The screenshot shows the Eclipse IDE interface with a code editor window titled "jaas.conf". The code defines a "Server" section with a single "Krb5LoginModule" entry. The module is configured with "useKeyTab=false", "storeKey=true", "useTicketCache=false", and a "principal" set to "webserver/bully@eccouncil.org". The code editor has a red border around the configuration section. The status bar at the bottom indicates "Writable" and the time "12:46".

Secured Kerberos Implementation (Cont'd)



- Implementing Kerberos with ***CallbackHandler.java**
- LoginCallbackHandler implements **CallbackHandler**

```
1 package Eccouncil.app.gss;
2 import javax.security.auth.callback.Callback;
3
4 /**
5 * Password callback handler for resolving password/usernames for a JAAS login.
6 * @author Amrit
7 */
8 public class LoginCallbackHandler implements CallbackHandler {
9
10    public LoginCallbackHandler() {
11        super();
12    }
13    private String CompanyId;
14    private char[] CompanyPasswd;
15    public void handle(Callback[] callbacks)
16        throws java.io.IOException, UnsupportedCallbackException {
17        for (int i = 0; i < callbacks.length; i++) {
18            if (callbacks[i] instanceof NameCallback) {
19                NameCallback Ncb = (NameCallback)callbacks[i];
20                Ncb.setName(CompanyId);
21            } else if (callbacks[i] instanceof PasswordCallback) {
22                PasswordCallback Pcb = (PasswordCallback)callbacks[i];
23                Pcb.setPassword(CompanyIdPasswd);
24            } else {
25                throw new UnsupportedCallbackException(callbacks[i]);
26            }
27        }
28    }
29    public void setUserId(String userId)
30    {
31        CompanyId = userId;
32    }
33    public void setPassword(String pwd)
34    {
35        CompanyPasswd = pwd.toCharArray();
36    }
37}
```

The screenshot shows the Eclipse IDE interface with a code editor window titled "LoginCallbackHandler.java". The code defines a "LoginCallbackHandler" class that implements the "CallbackHandler" interface. It contains a constructor, a "handle" method that iterates through an array of callbacks to resolve a user ID and password, and methods to set the user ID and password. The code editor has a red border around the "handle" method block. The status bar at the bottom indicates "Writable" and the time "12:46".

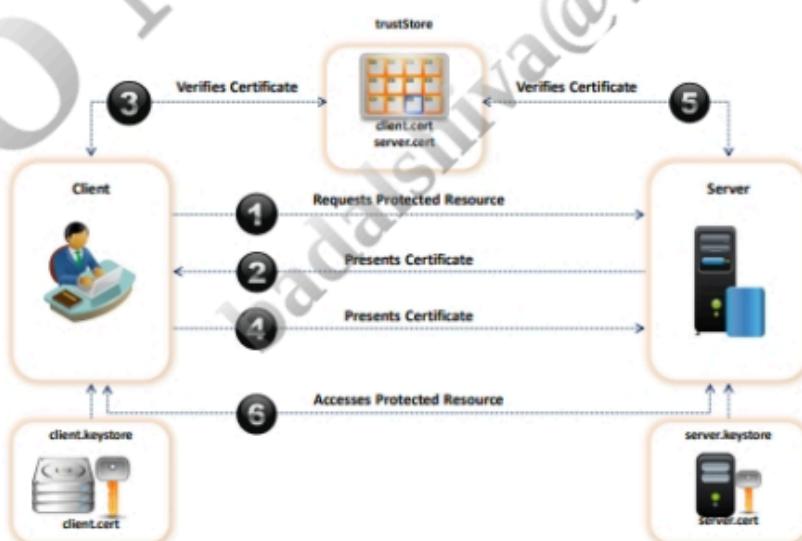
Client Certificate Authentication



- ❑ 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.

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 -keystore client.jks -storepass password
```

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.

Certificate Generation with Keytool (Cont'd)



Export the Server's Public Certificate and import it in to the Client's Keystore

```
C:\WINDOWS\system32\cmd.exe
C:\>keytool -exportcert -alias serverkey -file server-public.cer -keystore server.jks -storepass password

C:\WINDOWS\system32\cmd.exe
C:\>keytool -importcert -keystore client.jks -alias servcert -file server-public.cer -storepass password -noprompt

# view the contents of the keystore (use -v for verbose output)

C:\WINDOWS\system32\cmd.exe
C:\>keytool -list -keystore client.jks -storepass password
```

Certificate Generation with Keytool (Cont'd)



Configure Tomcat for SSL using the Server Keystore

```
<Connector
    clientAuth="true" port="8443" minSpareThreads="5" maxSpareThreads="75"
    enableLookups="true" disableUploadTimeout="true"
    acceptCount="100" maxThreads="200"
    scheme="https" secure="true" SSLEnabled="true"
    keystoreFile="/Users/mporges/Desktop/tomcat-ssl/final/server.jks"
    keystoreType="JKS" keystorePass="password"
    truststoreFile="/Users/mporges/Desktop/tomcat-ssl/final/server.jks"
    truststoreType="JKS" truststorePass="password"
    SSLVerifyClient="require" SSLEngine="on" SSLVerifyDepth="2" sslProtocol="TLS"
    />
```

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

Implementing Encryption and Certificates in Client Application



The screenshot shows the Eclipse IDE interface with the following Java code in a file named ClientApp.java:

```
1 import org.apache.commons.httpclient.HttpClient;
2 import org.apache.commons.httpclient.URI;
3 import org.apache.commons.httpclient.methods.GetMethod;
4
5 public class ClientApp {
6     static
7     {
8         System.setProperty("javax.net.ssl.trustStore", "/Users/mporges/Desktop/tomcat-ssl/final/client.jks");
9         System.setProperty("javax.net.ssl.trustStorePassword", "password");
10        System.setProperty("javax.net.ssl.keyStore", "/Users/mporges/Desktop/tomcat-ssl/final/client.jks");
11        System.setProperty("javax.net.ssl.keyStorePassword", "password");
12    }
13
14    public static void main(String[] args) {
15
16        HttpClient client = new HttpClient();
17        GetMethod method = new GetMethod();
18        try {
19            method.setURI(new URI("https://localhost:8443", false));
20            client.executeMethod(method);
21            System.out.println(method.getResponseBodyAsString());
22        } catch (Exception e) {
23            e.printStackTrace();
24        }
25    }
26}
```

A red box highlights the code block from line 16 to line 21, which sets the URI to https://localhost:8443 and prints the response body.



Authentication Weaknesses and Prevention

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=0>
 - <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

Login

Username:

Password:

Remember Me

ERROR: Invalid username. Lost your password?

Indicates user ID is correct but password is not

Login

Username: **admin**

Password:

Remember Me

ERROR: The password you entered for the username admin is incorrect. Lost your password?

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

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.

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**

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

- Defining **roles** to users
- Define **security roles** of an application to roles defined in memory realm
- Define the **resource** and **HTTP** method constraints for an application in web.xml file
- Apply the constraints that are accessible by role
- Check for user **authentication** for the application
- Allow the server to **process** the **request** when the details are verified

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  23
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  23
25
26 if(req.isUserInRole("Admin"))
27 {
28     System.out.println("welcome");
29 }
```

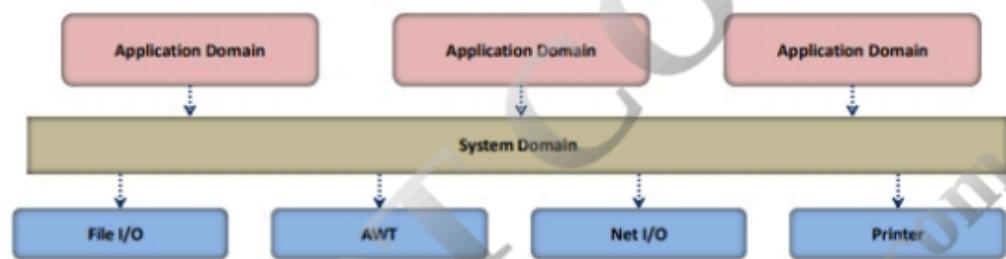


Access Control Model

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)



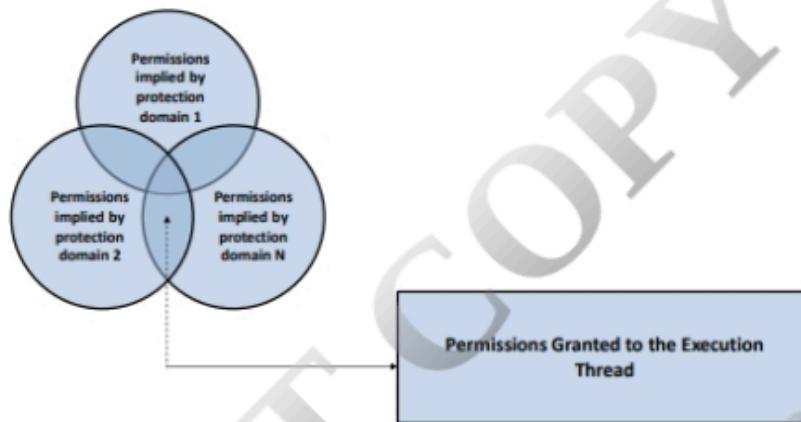
- The `AccessController.checkPermission()` method logic ensures each class in the thread stack gets `ProtectionDomain`
- All these `ProtectionDomains` are composed into a `java.security.AccessControlContext` object
- `implies()` method is invoked along with permission object on the `ProtectionDomain` 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.

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



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)

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

Mandatory Access Control (MAC)



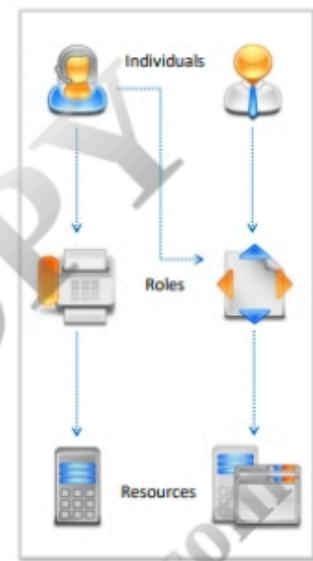
- This model relies on the principle of **limiting information dissemination**, and policies imposed by every data item in the system are governed by **Central Authority** throughout lifetime
- The creator of object (file) does not necessarily know who has **authorized access** to it
- The access control 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

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



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

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

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
```

```
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-collections>
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>meb</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 </!-->
67 </web-app>
```



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.

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.

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>
<assembly-descriptor>
<security-role>
<role-name>clerk</role-name>
</security-role>
<security-role>
<role-name>bob</role-name>
</security-role>
<method-permission>
<role-name>clerk</role-name>
<role-name>bob</role-name>
</method-permission>
<method>
<ejb-name>accountsPayable</ejb-name>
<method-name>getReceipts</method-name>
</method>
</method-permission>
<!-- ... -->
</assembly-descriptor>
</ejb-jar>

<!-- weblogic-ejb-jar.xml entries -->
<security-role-assignment>
<role-name>clerk</role-name>
<principal-name>ec</principal-name>
<principal-name>John</principal-name>
<principal-name>Jacob</principal-name>
</security-role-assignment>
```

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, `WLSUserPrincipal` is returned by method `javax.ejb.EJBContext.getCallerPrincipal()`
- If there are multiple `WLSUserPrincipal`s, the first one in `Subject.getPrincipals().iterator()` method is returned
- In case there are no `WLSUserPrincipal`s, the first non-`WLSGroupPrincipal` will be returned by method `getCallerPrincipal()`
- If principals are not present or all principals belong to `WLSGroup`, the method returns `weblogic.security.WLSPrincipals.getAnonymousUserPrincipal()`

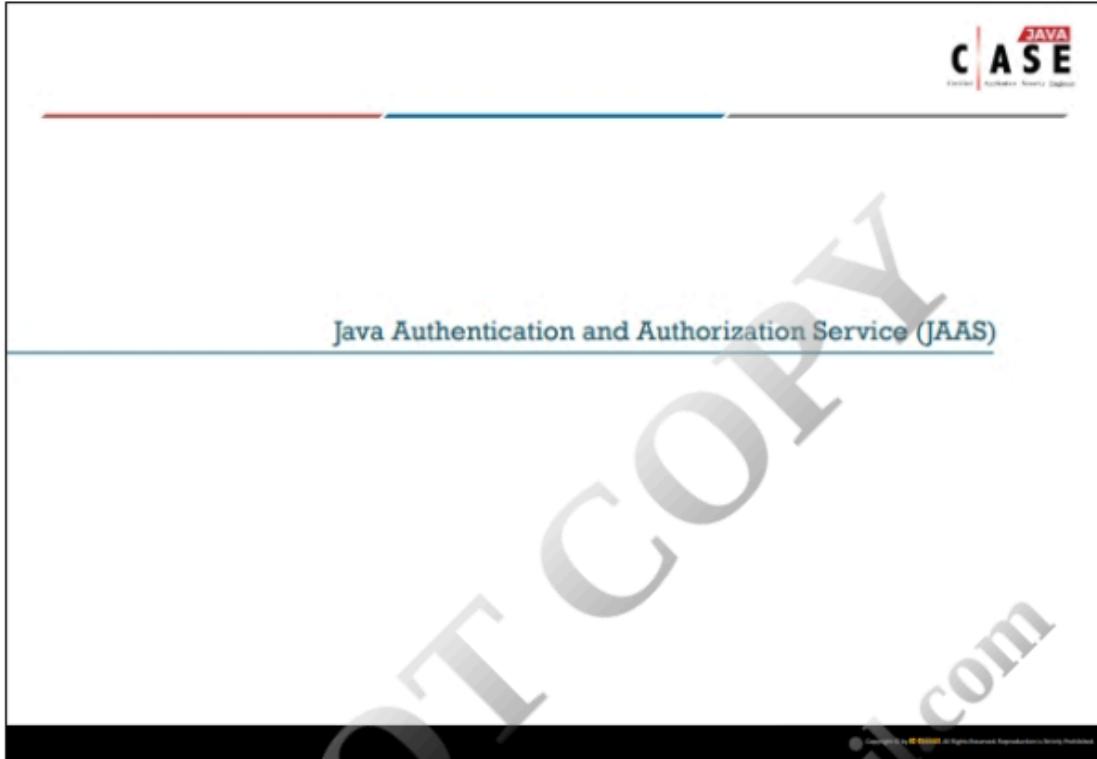
isCallerInRole

- 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

Java Authentication and Authorization Service (JAAS)

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

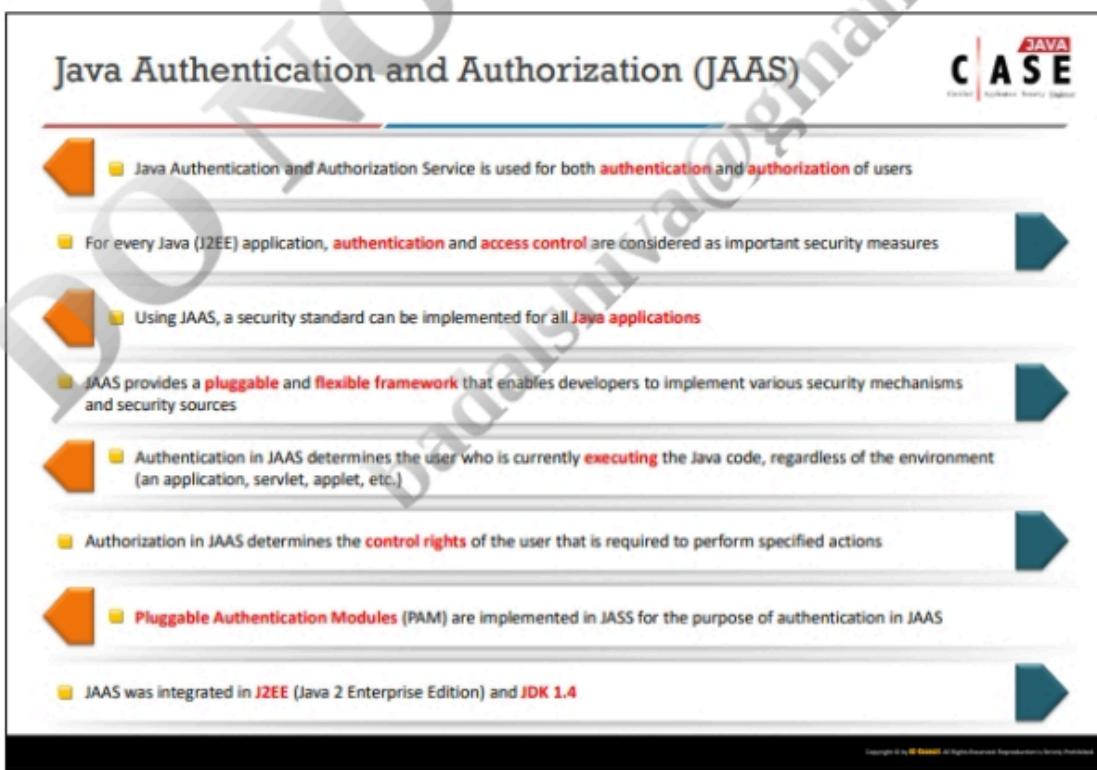


Java Authentication and Authorization (JAAS)

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

- 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.



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



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



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.