

## Logging in Java



- In Java, logging frameworks are provided by the package `java.util.logging` and used to write log messages on to a central location
  - The log messages can be `error messages`, `informational messages` or `configuration messages`

**Logging is divided into three parts:**

Logger	<ul style="list-style-type: none"><li>• Receives the message that is then passed to the <b>logging framework</b></li><li>• The framework then calls the <b>formatter</b> including the message</li></ul>
Formatter	<ul style="list-style-type: none"><li>• Formats the <b>message</b> for output</li><li>• The framework then <b>passes</b> the <b>output</b> to the handler</li></ul>
Handler (Appender)	<ul style="list-style-type: none"><li>• The appropriate <b>handler dispositions</b> the output</li><li>• Examples of Java Logging Frameworks:<ul style="list-style-type: none"><li>• Apache Commons Logging</li><li>• Java Logging API</li><li>• Log4J</li><li>• SLF4J</li></ul></li></ul>

## Example for Logging Exceptions



- The code below gives an example of logging an exception to a log file

The screenshot shows an IDE window titled "Invoke.java" with the following Java code:

```
7     Logger.getLogger(SimpleClass.class.getName());
8     public void writeLog() {
9         logger.info("Enable logging...");
10        try {
11            throw new Exception("Simulating an exception");
12        } catch (Exception e) {
13            logger.log(null, e.getMessage());
14        }
15    }
16    public static void main(String[] args) {
17        SimpleClass simpleObj = new SimpleClass();
18        simpleObj.writeLog();
19        logger.info("Done...");
20    }
21 }
```

Annotations are present in the code:

- A red box highlights the line `logger.log(null, e.getMessage());`.
- A red box highlights the line `logger.info("Done...");`.

## Logging Levels



Logging plays a major role in **web applications** as it is the only way to track the **server's activities**. Logging levels based on Java log4j are as follows:

### DEBUG

- The lowest restricted Java logging level
- The messages logged in this level are used to debug the application
- It is used only in the development and testing environment

### WARN

- This level is more restricted than the INFO level
- The messages logged in this level are used to give warning messages
- E.g. Connection lost between client and server, Database connection lost etc.

### FATAL

- This level indicates severe error conditions that leads the application to abort
- After this, the application may eventually crash or get stopped

### INFO

- This level is more restricted than the DEBUG Java logging level
- The messages logged in this level are used to give information to the programmers
- E.g. The server has started, incoming and outgoing messages

### ERROR

- This level is more restricted than the WARN level
- Error and exception messages are logged in this level
- ERRORS should always be printed in Java

### OFF

- This level has the highest possible rank and it is intended to turn off logging in Java

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



## Logging using Log4j

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

## Log4j and Java Logging API



	log4j	java.util.logging
Manager Class	org.apache.log4j.LogManager	java.util.logging.LogManager
Logger Objects	org.apache.log4j.Logger	java.util.logging.Logger
Named Loggers	Supported	Supported
Logging Levels	<ul style="list-style-type: none"><li>• Levels are declared in the org.apache.log4j.Level class</li><li>• log4j has five pre-defined levels:<ul style="list-style-type: none"><li>■ FATAL</li><li>■ ERROR</li><li>■ WARN</li><li>■ INFO</li><li>■ DEBUG</li></ul></li></ul>	<ul style="list-style-type: none"><li>• Levels are declared in the java.util.logging.Level class</li><li>• There are seven pre-defined levels:<ul style="list-style-type: none"><li>■ SEVERE</li><li>■ WARNING</li><li>■ INFO</li><li>■ CONFIG</li><li>■ FINE</li><li>■ FINER</li><li>■ FINEST</li></ul></li></ul>
Log Event Object	org.apache.log4j.spi.LoggingEvent	java.util.logging.LogRecord
Output Destinations	Appender classes - these classes implement the org.apache.log4j.Appender interface	Handler classes - these classes extend java.util.logging.Handler

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

## Log4j and Java Logging API (Cont'd)



	log4j	java.util.logging
Output Message Filtering	Filter classes. These classes extend org.apache.log4j.spi.Filter	Filter classes. These classes must implement the java.util.logging.Filter interface
Output Message Formatting	Layout classes extend org.apache.log4j.Layout	Formatter classes extend java.util.logging.Formatter
Programmatic Configuration	Supported	Supported
Property File Configuration	Supported	Supported
XML Configuration	Supported. The XML configuration file must comply with log4j.dtd	Not supported
Email Notification	SMTPAppender class	SMTPHandler class (third-party extension)

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

## Java Logging using Log4j



Given below is a sample code used to log files using Log4j

```
1 *Invoke.java *
2
3 import org.apache.log4j.Logger;
4
5 public class Invoke {
6     private static final Logger logger =
7         Logger.getLogger(SimpleClass.class.getName());
8     public void writeLog() {
9         logger.debug("Here Debug message");
10        logger.info("Enable logging...");
11        logger.warn("Warning Message");
12        logger.error("Here Error message");
13        logger.fatal("fatal error message");
14
15    try {
16        throw new Exception("Simulating an exception");
17
18    } catch (Exception e){
19        logger.log(null, e.getMessage());
20    }
21 }
22
23 public static void main(String[] args) {
24
25     SimpleClass simpleObj = new SimpleClass();
26     simpleObj.writeLog();
27     logger.info("Done...");
28
29 }
30 }
```

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



## Secure Coding in Logging

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

## Vulnerabilities in Logging



1. It is **not a good practice** to log and then throw the errors

```
1# import java.io.BufferedReader;
2
3 public class DemoForException {
4     private static final Logger logger =
5         Logger.getLogger(SimpleClass.class.getName());
6     public void test() {
7         try {
8             int i=5;
9             i=i/0;
10        } catch (Exception ex) {
11            Logger.error("Error",ex);
12            throw ex;
13        }
14    }
15 }
```

2. Either log exceptions or throw exceptions; both **should not be done** together

Example: Log Exception

```
26 public void Info() throws MyException {
27 {
28     try {
29         int i=5;
30         i=i/0;
31     } catch (ArithmeticException ex) {
32         logger.error("Error",ex);
33         throw new MyException("Error",ex);
34     }
35 }
```

Example: Throw Exception

```
38 try {
39
40 }
41 catch (ArithmeticException ex) {
42
43     ex.printStackTrace();
44     throw new MyException("Error",ex);
45 }
46 }
```

## Vulnerabilities in Logging (Cont'd)



3. Exceptions should be **thrown** instead of returning null values are returned only in a normal use case

```
47     try {
48
49     }
50     catch (ArithmeticException ex) {
51
52         ex.printStackTrace();
53         return;
54     }
55 }
```

4. Null values are returned only in a **normal use case**

```
22 public String map()
23 {try{
24
25 }
26 catch(Exception e)
27 {
28     return null;
29 }}
```

## Logging: Vulnerable Code and Secure Code

**1. Group log messages regardless of the level**

The vulnerable code below may look fine in a test case, but when it is displayed in the log file of an application server with 500 threads running parallel, the two messages may be spaced up with 1000 lines in-between in the log file

**Vulnerable Code**

```
try {
    Logger.debug("Using cache policy A");
    Logger.debug("Using retry policy B");
}
catch (ArithmeticException ex) {
```

**Secure Code**

```
try {
    Logger.debug("Using cache policy A, using retry policy B");
}
catch (ArithmeticException ex) {
```

## Logging: Vulnerable Code and Secure Code (Cont'd)

**2. Parameterized logging using SLF4J**

**Vulnerable Code**

```
if(logger.isDebugEnabled())
{
    logger.debug("created user[" + user + "] with role [" + role + "]");
}

try {
    logger.error("message",ex);
}
catch (ArithmeticException ex) {
```

**Secure Code**

```
try {
    slf4jLogger.debug("created user[{}] with role [{}]",user,role);
}
catch (ArithmeticException ex) {
    slf4jLogger.error("message",ex);
}
```

## Logging: Vulnerable Code and Secure Code (Cont'd)

### 3. Log the exception stacktrace

#### Vulnerable Code

```
57     }
58
59     try {
60         int i=5;
61         i=i/0;
62     }
63     catch (ArithmaticException ex) {
64         logger.error("Error in application"+ex);
65     }
66
67 }
68
69 }
```

#### Secure Code

```
57     }
58
59     try {
60         int i=5;
61         i=i/0;
62     }
63     catch (ArithmaticException ex) {
64         logger.error("Error in Application", ex);
65     }
66
67 }
68
69 }
```

## Logging: Vulnerable Code and Secure Code (Cont'd)

### 4. Log exceptions in proper order

#### Vulnerable Code

```
33 {
34     try {
35         int i=5;
36         i=i/0;
37     } catch (Exception lolevelException) {
38
39
40         throw new ArithmaticException();
41     }
42 }
```

#### Secure Code

```
36     int i=5;
37     i=i/0;
38 } catch (Exception lolevel) {
39
40         throw new HighLevelException("Error", lolevel);
41     }
42 }
```

## Logging: Vulnerable Code and Secure Code (Cont'd)



5. Log exceptions when they are handled. Do not log messages before they are handled

### Vulnerable Code

```
35 try {
36     int i=5;
37     i=i/0;
38 } catch (Exception hklvel) {
39
40     slf4jLogger.error("catch Error herer", hklvel);
41 }
```

## Logging: Vulnerable Code and Secure Code (Cont'd)



6. Log uncaught exceptions

### Vulnerable Code

- In web applications, uncaught exceptions can be sent to a **JSP page** and log it as follows:

```
<error-page>
<exception-type>EmployeeNotFoundException</exception-type>
<location>Sorry.jsp</location>
```

### Secure Code

- The example code is as follows:

```
12 public static void main(String args[]) throws IOException
13 {
14
15     Thread.setDefaultUncaughtExceptionHandler(new UncaughtExceptionHandler{
16         @Override
17         public void uncaughtException(Thread t, Throwable e) {
18             Logger log = org.slf4j.LoggerFactory.getLogger(t.getClass());
19             log.error("Exception in thread [" + t.getName() + "]", t);
20         }
21     });
22 }
```



## Secured Practices in Logging

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

## Secured Practices in Logging



- When logging, as there is an increase in file **IO operations**, it affects the performance of the application very badly
- The following best practices should be implemented for effective Java logging:
  - Log DEBUG messages inside **isDebugEnabled()** block as given below:

The screenshot shows a Java code editor window with a file named "Invoke.java". The code is as follows:

```
17
18
19
20
21
if(Logger.isDebugEnabled())
{
    logger.debug("Debug Log Enabled");
}
```

- This practice saves a lot of string concatenation activity if the code is running in the production environment with **production logging level** instead of **debug logging level**
- Make use of good Java logging frameworks like **java.util.logging** or **log4j**
- For example, by using **Log4j** the logging levels can be changed without restarting the application and that gives a great advantage in a production or controlled environment

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

## Secured Practices in Logging (Cont'd)



- Do not log sensitive data when logging in a **production environment**. Logging sensitive information may cause security risks.  
Example: e-mail address, phone numbers and account numbers should not be logged

- In server applications, all **bugs** and **errors** can be **tracked** only by using Java logs and too many logs will affect the application; therefore, select the appropriate logging level for every logging message

- Ensure that the format of the Java logging in the specified Java **logger** is included. In the case of **java.util.logging API**, the properties file should include the **logging formatter**. While printing logs the thread name and the Java class name should be specified

- Log messages consistently and the messages must be **informative**. The data should be **printed** with the **message** wherever required

- Use code level prefix when logging information (e.g. **database code** or **session code** etc.) so that it will be clear to identify which part of the code is **printing** the log message

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

## Module Summary



- Exception and error occur when an unexpected or abnormal event takes place during the execution of the application
- Exceptions in Java are handled using 'try-catch-finally' blocks
- `java.lang.InterruptedIOException` is thrown when a thread is interrupted while sleeping or waiting
- Use of return, break, continue or throw statements is restricted in a finally block
- `System.exit()` when invoked can even terminate the Java Virtual Machine (JVM) resulting in the termination currently running programs as well as threads that might even lead to denial-of-service (DoS) attacks
- `RuntimeException`, `Exception`, or `Throwable` should not be thrown or else it may lead to various errors
- `printStackTrace()` function should not be used in code as it reveals all the exception information to the user
- In Java, logging frameworks are provided from package `java.util.logging` and used to write log messages on to a central location

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



A background graphic featuring a green circuit board pattern. Overlaid on the board is a large, semi-transparent white stamp that reads "TESTED" in bold, sans-serif capital letters. A faint watermark reading "DO NOT COPY" is also visible across the stamp. The overall aesthetic is technical and modern.

## Module 09

# Static and Dynamic Application Security Testing (SAST & DAST)

DO NOT COPY  
This page is intentionally left blank.  
[badalshiva@gmail.com](mailto:badalshiva@gmail.com)

## Module Objectives



- 1 Introduction to Static Application Security Testing (SAST)
- 2 Discuss Manual Secure Code Review Techniques for Most Common Vulnerabilities
- 3 Introduction to Dynamic Application Security Testing
- 4 Discuss DAST using Automated Application Vulnerability Scanning Tools
- 5 Discuss DAST using Proxy-based Security Testing Tools

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



## Static Application Security Testing

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

## Static Application Security Testing (SAST)



- 1 SAST, also known as **secure code review**, is one of the software security assurance approach to identify security-related **weaknesses** in the code
- 2 It involves detailed systematic inspection of **source code** to detect vulnerabilities and design flaws
- 3 It should be performed toward the end of the source code development when application code is **stable** or **nearly completed**
- 4 It should always be performed in combination with **human effort (Manual)** and **technology support (Automated)**

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

## Objectives of SAST



- SAST aims to detect **application security vulnerabilities** and their **root causes** when code is not running

### Root cause analysis (Source to sink)

Source to Sink analysis (what is the source (input) and the sink (vulnerable code pattern) for any vulnerability) is carried out using **Data flow diagrams**

### Design Analysis

Design flaws like **mishandling of nonuser inputs** and external integrations like **server-to-server** can be easily identified

### Find all Instances of the Common Vulnerabilities

Security code review enumerates all instances of **certain vulnerability**

### Identify Uncommon Security Flaws

Security code review being specific to the application may **highlight some flaws** like insecure termination of execution flow, synchronization error, etc.

### Analyze Effectiveness of Existing Security Controls

Security code review enumerates **weakness** in the existing security control against known **threats**

### Effective Remediation Recommendations

Suggests **mitigation techniques** that can best suit the application instead of a generic one

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

## Why SAST



- Many serious **security vulnerabilities** can be **addressed**
- Decreases the remediation **cost of vulnerabilities** in later stages
- Deeper analysis is possible with **less false positive and negative**
- Help you to add best secure coding practices as per **department coding standards** which can prevent future security issues

## Skills Required for SAST



- 1 Able to think from the attacker's perspective
- 2 Working knowledge of the **programming language** under review
- 3 Knowledge of standard, secure coding techniques
- 4 Quickly able to **detect insecure coding** practices
- 5 Analytical skills

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

## What to Look for in SAST



- Look for the code that implements common application security mechanisms
- The **insecure** implementation of security mechanisms can render applications vulnerable to various attacks

### Common Application Security Mechanisms



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

## Common Vulnerabilities Identified through SAST



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

## Types of SAST



### Types Of SAST

#### Automated Source Code Analysis

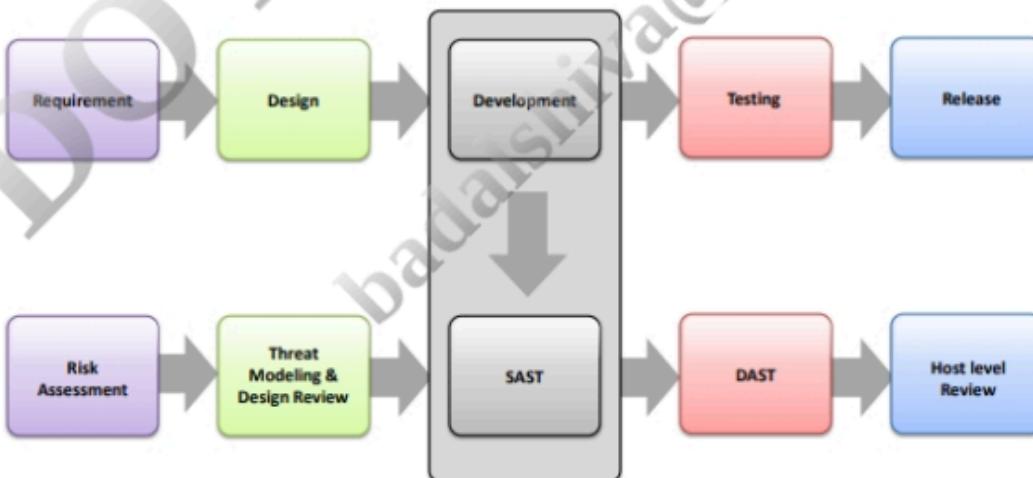
- It is also known as **Static Code Analysis (SCA)**
- It uses certain source code analysis tool to scan the code and **report potential flaws**

#### Manual Source Code Review

- It involves manually inspecting the source code line by line to detect any **defects** and **security related flaws**

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

## Where does SAST Fit in SDLC?



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

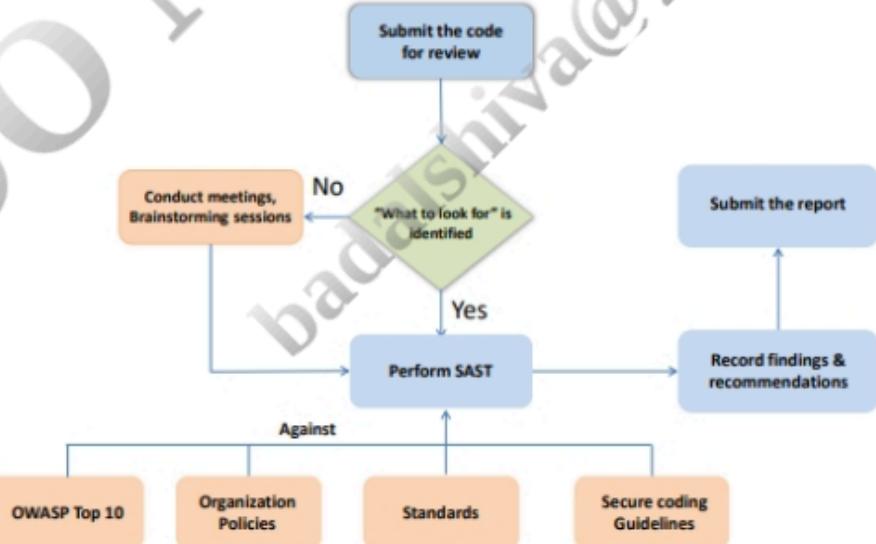
## SAST Steps



1. Perform an application walkthrough
  - Understand the applications functionality
2. Developers Interview
  - Talk to **developers**; more you involve developers, more effective your secure code review will be
  - It will help you quickly
    - Determine the **high risk areas**
    - Understand **developer's logic** behind particular functionality
    - Establish **friendly** and **respectful** relationship between you and developers
3. Analyze threat Model
  - Figure out the major existing **security threats**
4. Define secure code review objectives
  - Based on threat model analysis, identify the objectives such as:
    - Code review for all major **security vulnerabilities**
    - Code review for **OWASP top 10**
    - Code review for **specific vulnerability**
5. Define scope for secure code review
  - Be clear on what to **look for** and what to **avoid**
  - Calculate the **time** and **efforts** required
  - Define the **budget constraints**
6. Categorized the Objectives
  - Prioritize the objective based on **severity** and **requires more attention**
7. Conduct Static Code Analysis
  - Use **multiple** source code analysis tools to perform secure source code analysis
  - Most common vulnerabilities and flaws are **identified**
8. Conduct Manual Source code Review
  - Inspect the application's code line by line in order to find the **use of insecure coding practices**
9. Remediation/Recommendation
  - Suggest secure coding practice for each **findings**
10. Prepare Report

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

## SAST Activities-flow Chart



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

## Recommendation for Effective SAST



- |  |  |
|--|--|
| <b>01</b><br>Code review should always be done by a <b>third party person, a peer or an external entity</b>  | <b>07</b><br>Set a <b>reasonable time limit</b> for the review   |
| <b>02</b><br>Understand the application architecture and business logic and sort out the <b>possible security issues</b> that the application can be vulnerable to     | <b>08</b><br>Do not lose track while going into detail of the <b>higher-level</b> security vulnerabilities   |
| <b>03</b><br>Set clear set of objectives for " <b>what to look for</b> " based on security issues  | <b>09</b><br>Do not spend too much time in not a high-priority area or objective, <b>address them later</b>  |
| <b>04</b><br>Review code for security objectives only. Do not review code for other issues like performance, business logic, reliability, etc. Do it <b>separately</b> | <b>10</b><br>Always select small and manageable piece of code for review. It will help you to find a larger number of security vulnerabilities with <b>great focus</b> and within <b>less time</b> |
| <b>05</b><br>Spend time to set <b>clear objectives</b>   | <b>11</b><br>Use multi-pass approach. Let the multiple people allow to review the <b>same code</b>   |
| <b>06</b><br>Prioritize the security issues; higher-level security vulnerabilities should be <b>addressed first</b>  | <b>12</b><br>Detect secure coding mistakes that are repeatedly committed and add the corresponding secure coding practice in your <b>development department's coding standards</b>                 |

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

## SAST Deliverable



### SAST report should include:

- 1** Types of vulnerabilities
- 2** Severity level/Impact
- 3** Location, line of code where security issue exists
- 4** Recommendations

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

## Automated Source Code Analysis



- With the help of automated source code analysis, you can **identify** the code associated with most common application **security vulnerabilities**
- The **static code analysis** tools automate the entire process of secure code review; however, they scans the code **superficially**
- Multiple static code analysis tools should be used to conduct **secure code review**

### Advantages:

- Takes **very less time** to complete

### Disadvantages:

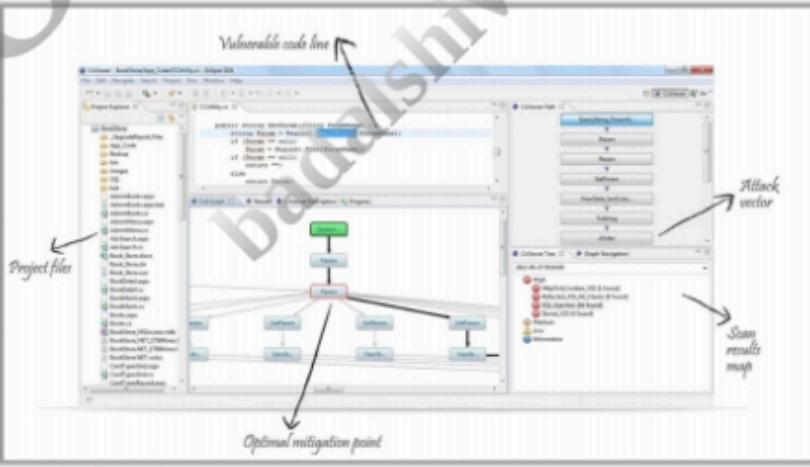
- Unable to **uncover** every issue
- Good automated review tools are **expensive**
- High possibility of **false-positive** or **false-negative errors**
- Requires **human intervention** to deal with of false-positive or false-negative

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

## Static Code Analysis using Checkmarx Static Code Analysis



- **Checkmarx** static code analysis software, seamlessly integrated into development process which identifies hundreds of known code vulnerabilities



Source: <https://www.checkmarx.com>

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

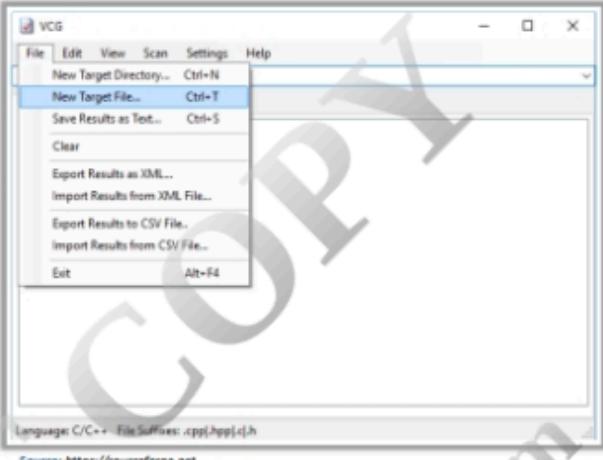
## Static Code Analysis using Visual Code Grepper (VCG)



VCG is an automated code security review tool for C++, C#, VB, PHP, Java and PL/SQL which is intended to drastically speed up the code review process by identifying bad/insecure code

Steps for SCA:

- Install and run the VCG tool
- Select the language of your project
- Browse the project/file of your interest and click Scan



Language: C/C++ File Suffixes: .cpp|.hpp|.c|.h

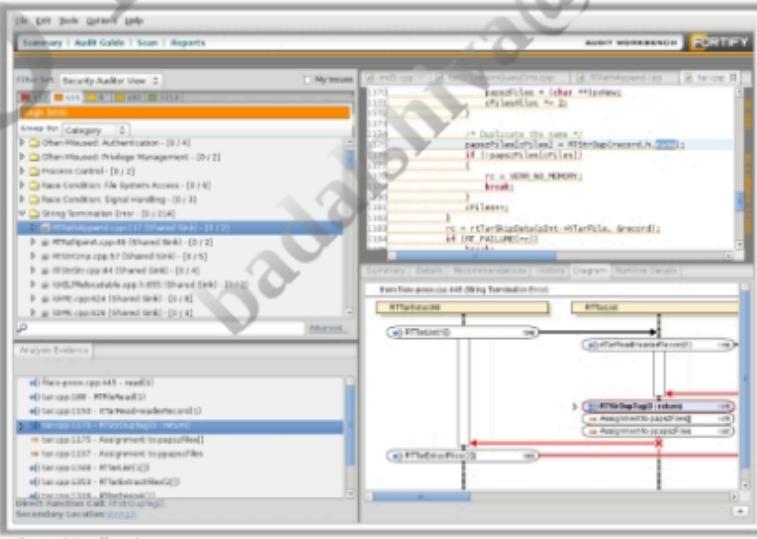
Source: <https://sourceforge.net>

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

## Static Code Analysis using HP Fortify



Fortify Static Code Analyzer is an automated static code analysis to help developers eliminate vulnerabilities and build secure software



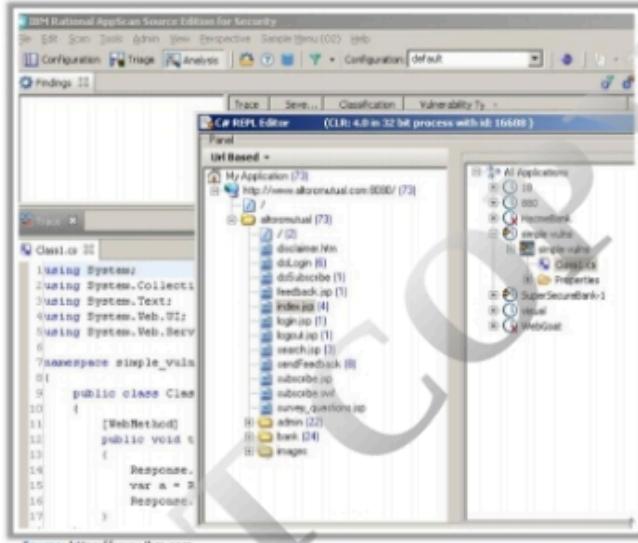
Source: <https://soas.hpe.com>

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

## Static Code Analysis using Rational AppScan Source Edition



- AppScan Source is a static application security testing (SAST) solution which scans your application source code for **security vulnerabilities**



Source: <https://www.ibm.com>

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

## Selecting a Static Analysis Tool



- You need to select the right tool while selecting the static analysis tool for secure code review
- The process of selecting the **right tool** depends on certain conditions:
  - The static analysis tool used for reviewing the code should support the **programming language** used for developing the application
  - The accuracy of the tool with respect to **false-negative** and **false-positive errors**
  - The tool needs to understand the **libraries** and **frameworks** used in the application
  - It should be easy to **integrate the tool** to the IDE selected for developing the application
    - The **time** and **effort** required for the set-up of the tool
    - The type of analysis preferred for code review process
    - The license cost incurred for the tool
- You should have a clear idea regarding the type of static code analysis required for performing **code review**

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

## Manual Secure Code Review



- With manual secure code review, you can analyze and detect many security vulnerabilities that are missed out during the static code analysis

### Advantages

- Less chance of getting false-positive and false-negatives
- High percentage of vulnerabilities can be detected

### Disadvantages

- Time consuming process
- Requires good understanding of the code used

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



## Manual Secure Code Review for Most Common Vulnerabilities

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

## Code Review for PCI DSS Compliance



- The PCI DSS compliancy is checked against a list of **PCI DSS requirements**
- Check the applications against a list of PCI DSS requirements

PCI DSS Requirement	Description
6.3.7	Review of the custom code for vulnerabilities before releasing the code to production or customers
6.3.5	Deletion of custom user accounts, usernames and passwords before releasing it to the customers
6.5	Review and prevention of common coding vulnerabilities in an application
6.6	Review of custom application codes and installation of application layer firewall for web applications

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

## Code Review for PCI DSS Compliance (Cont'd)



- The **PCI DSS requirement 6.5** reviews and prevents common coding vulnerabilities such as:

<input type="checkbox"/> 6.5.1 Unvalidated input	<input type="checkbox"/> 6.5.6 Injection flaws
<input type="checkbox"/> 6.5.2 Broken access control	<input type="checkbox"/> 6.5.7 Improper error handling
<input type="checkbox"/> 6.5.3 Broken authentication and session management	<input type="checkbox"/> 6.5.8 Insecure storage
<input type="checkbox"/> 6.5.4 Cross-site scripting (XSS) attacks	<input type="checkbox"/> 6.5.9 Denial of service
<input type="checkbox"/> 6.5.5 Buffer overflows	<input type="checkbox"/> 6.5.10 Insecure configuration management

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

## Code Review for Blacklisting Validation Approach



Code Pattern: Look for code where the developer is attempting to use a **blacklisting approach** to validate user inputs

```
Source Navigate Search Project Run Window Help
*Login.jsp  ValidateUser.java
1 package Validation;
2
3 public class ValidateUser {
4
5
6 int valiuser()
7 {
8     String[] userInputs= new String[]{"<SCRIPT>","SELECT","UNION","WHERE","</SCRIPT>","--",">","<"}
9     String userName =getUserName();
10    if(userName.toUpperCase().contains(userInputs.toString()))
11    {
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
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
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
```

## Code Review for Non-parametrized SQL Query

Code Pattern: Look for code where the developer is attempting to write a **non-parametrized query**

The screenshot shows a Java code editor with the file 'LoginController.java' open. The code attempts to connect to an Oracle database and execute a non-parametrized SQL query:

```
String uname = request.getParameter("uname");
String pwd = request.getParameter("pwd");
int flag = -1;
try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "system", "admin");
    PreparedStatement ps = con.prepareStatement("select * from empinfo where uname='"+uname+"' and pwd='"+pwd+"'");
    ResultSet rs = ps.executeQuery();
}
```

A red box highlights the non-parametrized SQL query: `PreparedStatement ps = con.prepareStatement("select * from empinfo where uname='"+uname+"' and pwd='"+pwd+"'");`

Recommendation: Always use **parametrized query** to avoid **SQL injection attacks**

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

## Code Review for XSS Vulnerability

Code Pattern: Look for code where the developer is attempting to **render output without encoding**

The screenshot shows a Java code editor with the file 'FetchMatching.java' open. The code attempts to print user input directly to the console without encoding:

```
try{
    String dt=request.getParameter("text");
    out.println("You message ....\"+dt+" has been send");
} catch(InvalidArgumentException e)
{
    System.out.println(e);
}
```

A red box highlights the unencoded output line: `out.println("You message ....\"+dt+" has been send");`

Recommendation: Always **encode** your output to avoid **XSS**

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

## Code Review for Weak Password Authentication



Code Pattern: Look for code where the developer is attempting to authenticate users with a **weak password condition**

```
File Edit Source Refactor Navigate Search Project Test Run Window Help
CheckLogin.jsp
44
45     String password =request.getParameter("pass");
46
47     if(password.isEmpty())
48     {
49         throw new NullPointerException();
50     }
51     else {
52         response.sendRedirect("Home.jsp");
53     }
54
55
56
57
58
59
60
61
62
```

Design Preview Writable Smart Insert 44:1

html/body/section/div/jspscriptlet/Root

Recommendation: Always use **strong password rules** or **conditions** to authenticate users

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

## Code Review for Hard-coded Passwords



Code pattern: Look for code where the developer is attempting to use **hard-coded passwords**

```
File Edit Source Refactor Navigate Search Project Test Run Window Help
CheckLogin.jsp
47
48     if (user.equals("user") && pwd.equals("pass"))
49     {
50
51         string = response.encodeURL("Home.jsp?user=" + user + "&pwd=" + pwd + "");
52
53         %> <a href="<%>string<%>"> class="btn-get-started">Get Started</a>
54     } else {
55
56         response.sendRedirect(response.encodeRedirectURL("Login.jsp"));
57     }
58 } catch (Exception e) {
59     e.getMessage();
60 }
61
62
```

Recommendation: Never use hard-coded secrets such as **passwords** in coding. Hard-coded passwords can be an easy target for attackers

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

## Code Review for Empty Password in Connection String



**Code pattern:** Look for the connection string where the developer is attempting to use an **empty password** to connect to a remote database server

```
String uname = request.getParameter("uname");
String pwd = request.getParameter("pwd");
int flag = -1;

try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "system", "");
    PreparedStatement ps = con
        .prepareStatement("select * from ecempinfo where uname='"
            + uname + "' and pwd='"
            + pwd + "'");
    ResultSet rs = ps.executeQuery();

    boolean status = false;
    status = rs.next();
    if (status) {
        %>
        <jsp:include page="index.jsp" />
        <%
    } else
    {
    }
}
```

**Recommendation:** Never set **empty or plain text** for connection string as this will facilitate **dictionary attacks or brute force attacks**

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

## Code Review for Insecure Basic Authentication



**Code Pattern:** Look for code where the developer is attempting to use **basic authentication and HTTP connection** without using SSL

```
import org.apache.tomcat.util.codec.binary.Base64;
public class ValidateUser {
    String username="test";
    String password="pass";
    void getConnection() throws IOException {
        String targetServer="show.jsp";
        URL url = new URL(targetServer);
        HttpURLConnection conn = (HttpURLConnection)url.openConnection();
        String creds = username + ":" + password;
        String basicAuth = "Basic " + new String(new Base64().encode(creds.getBytes()));
        conn.setRequestProperty ("Authorization", basicAuth);
    }
}
```

**Recommendation:** Use **SSL** for basic authentication so that credential can not be decoded easily

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

## Code Review for Open Redirect



Code Pattern: Look for code where the developer is attempting to **redirect** to the path that is created from user supplied input

```
File Edit Source Refactor Navigate Search Project Test Run Window Help
File Navigator Properties Servers Data Source Explorer Snippets Console REST Annotations WebLogic Web Service Annotations
Quick Access
Test.jsp
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2 pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4<html>
5<head>
6 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7 <title>User Information</title>
8 </head>
9<body>
10<%
11 String url=request.getParameter("param");
12 response.sendRedirect("Info.jsp?param"+url);
13 %>
14
15 </body>
16 </html>
```

Recommendation: Always avoid using **redirects** or **forwards**. User input should not be used to construct the source path. If it is necessary to send a parameter then check if the parameter is valid. Ensure that the user is authorized to browse the constructed URL

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

## Code Review for Insecure LDAP Authentication



Code Pattern: Look for code where the developer is attempting to use "**simple**" basic authentication which is weak; attackers can easily get cleartext

```
File Edit Source Refactor Navigate Search Project Test Run Window Help
File Navigator Properties Servers Data Source Explorer Snippets Console REST Annotations WebLogic Web Service Annotations
Writable Smart Insert 20:27
ValidateUser.java
27
28
29
30
31 env.put(Context.PROVIDER_URL, "ldap://192.168.25:108");
32
33 env.put(Context.SECURITY_AUTHENTICATION, "simple");
34 DirContext ctx = new InitialDirContext(env);
35 SearchControls getCtlis = new SearchControls();
36 String returnedAttrs[] = {"address", "info", "searchname", "AreaNo", "EmpId", "uid"};
37 getCtlis.setReturningAttributes(returnedAttrs);
38 getCtlis.setSearchScope(SearchControls.SUBTREE_SCOPE);
39 String getFilter = filter;
40 String getBase = "ou=people,dc=mycompany,dc=com";
41 Hashtable<String, String> env = new Hashtable<String, String>();
```

Recommendation: Always use secure mechanisms like those listed in LDAP. LDAP supports **SASL** mechanism, SASL provides a security layer for authentication between client and server

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

## Code Review for Insecure Authorization Mechanism

**Code Pattern:** Look for code where the developer is attempting to use an **inadequate authorization mechanism** by which the code becomes complex and also hard to maintain. Even if the mistake is simple, an attacker can easily abuse the code

The screenshot shows a Java IDE interface with a JSP file named 'Test.jsp'. The code contains several conditional statements using the `request.isUserInRole()` method to check user roles. Lines 11 and 14 are highlighted with red boxes, indicating they are part of the 'inadequate authorization mechanism' being reviewed.

```
File Edit Source Refactor Navigate Search Project Text Run Window Help
Test.jsp
4<html>
5<head>
6<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7<title>User Information</title>
8</head>
9<body>
10<
11 if (request.isUserInRole("admin")){
12     response.sendRedirect("userInfo.jsp");
13 }
14 else if (request.isUserInRole("user")){
15     response.sendRedirect("userInfo.jsp");
16 }
17 else {
18 }
```

**Recommendation:** Always use a **proper authorization** method which is secure and easy to be altered

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

## Code Review for Weak Password Length

**Code pattern:** Look for code where the developer is attempting to set passwords with a **weak hashing** algorithm

The screenshot shows a Java IDE interface with a JSP file named 'BackendReg.jsp'. The code contains a method named `SimpleGeneratedPassword` that uses SHA-1 hashing. Line 181 is highlighted with a red box, indicating it is part of the 'weak hashing algorithm' being reviewed.

```
File Edit Source Refactor Navigate Search Project Text Run Window Help
BackendReg.jsp
95 } %>
96 
97@private static String SimpleGeneratedPassword(String password){
98 String passwordToHash = password;
99 String generatedPassword = null;
100 try {
101     MessageDigest md = MessageDigest.getInstance("SHA1");
102     md.update(passwordToHash.getBytes());
103     byte[] bytes = md.digest();
104     StringBuilder sb = new StringBuilder();
105     for(int i=0; i< bytes.length ;i++) {
106         sb.append(Integer.toString((bytes[i] & 0xff) + 0x100, 16).substring(1));
107     }
108     generatedPassword = sb.toString();
109 }
110 catch (NoSuchAlgorithmException e)
111 {
112     e.printStackTrace();
113 }
```

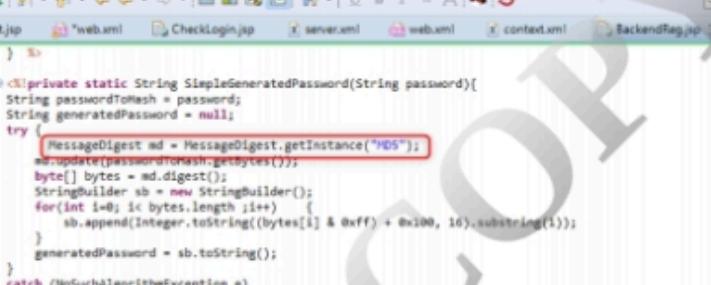
**Recommendation:** Set passwords with a strong hashing function like **Bcrypt** to make it unbreakable

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

## Code Review for Use of Weak Hashing Algorithm



■ **Code Pattern:** Look for code where the developer is attempting to set a **weak hashing algorithm**.



The screenshot shows the Eclipse IDE interface with the Java perspective active. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Test, Run, Window, and Help. The toolbar contains various icons for file operations like Open, Save, and Cut. The left sidebar shows the package structure with 'Test.jsp' selected. The main editor area displays Java code for generating MD5 hashes:

```
95 } >
96
97 private static String SimpleGeneratedPassword(String password){
98     String passwordToHash = password;
99     String generatedPassword = null;
100    try {
101        MessageDigest md = MessageDigest.getInstance("MD5");
102        md.update(passwordToHash.getBytes());
103        byte[] bytes = md.digest();
104        StringBuilder sb = new StringBuilder();
105        for(int i=0; i<bytes.length ; i++) {
106            sb.append(Integer.toHexString((bytes[i] & 0xff) + 0x100), 16).substring(1);
107        }
108        generatedPassword = sb.toString();
109    }
110    catch (NoSuchAlgorithmException e)
111    {
112        e.printStackTrace();
113    }
}
```

**Recommendation:** Always use **strongest hashing algorithm**

Copyright © by Houghton Mifflin Company. All rights reserved. Reproduction is strictly prohibited.

# Code Review for Use of Weak Random Number Generator



- Code Pattern: Look for code where the developer is using a **weak random number generation method**. An 8 character string can be predicted by an attacker

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

- Title Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Java development tools like New Project, Open, Save, Find, Replace, etc.
- Quick Access:** A search bar labeled "Open Task (Ctrl+F12)" and a "Quick Access" button.
- Code Editor:** The code for `ValidateUser.java` is displayed. The code generates a random string from a pool of symbols.

```
String symbols = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
Random random = new Random();
char[] buffer = new char[8];
for (int i = 0; i < buffer.length; ++i)
{
    buffer[i] = symbols.charAt(random.nextInt(symbols.length()));
}
String randomString = new String(buffer);
```
- Right Panel:** Shows icons for Files, Projects, and Servers.
- Bottom Navigation:** Markers, Properties, Servers, Data Sour..., Snippets, Console, REST Ann..., WebLogic..., Writable, Smart Insert, 36:5.

**Recommendation:** To prevent an attacker from predicting random numbers, use a string with more than **8 characters** to generate random numbers

Downloaded by [128.111.121.42] at 06:45 04 September 2016

## Code Review for Use of Insecure PBE Work Factor



Code Pattern: Look for code the developer is using with a work factor of 5000 which is considered to be insecure

```
File Edit Source Refactor Navigate Search Project Run Window Help
ValidateUser.java Login.jsp
public void userinfo()
{
    String password="";
    String salt="";
    try {
        char[] passwordChars = password.toCharArray();
        byte[] saltBytes = salt.getBytes();
        PBEKeySpec spec = new PBEKeySpec(passwordChars, saltBytes, 5000, KEY_LENGTH);
        SecretKeyFactory key = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
        byte[] hashedPassword = key.generateSecret(spec).getEncoded();
    }
}
Markers Properties Servers Data Sources Support Console REST Ann... WebLogic...
Writable Smart Insert 38:5
```

Recommendation: The developer should use big iteration value which generates complex hashing to prevent the password from being predicted

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

## Code Review for Use of Weak Encryption Algorithm



Code Pattern: Look for code where the developer is attempting to set a weak encryption algorithm

```
File Edit Source Refactor Navigate Search Project Run Window Help
ValidateUser.java
package Validation;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
public class ValidateUser {
    public static String userDataEncrypt(String userClearText, String userKey) throws Exception{
        String userInfo="";
        try {
            SecretKeySpec skypespec=new SecretKeySpec(userKey.getBytes(),"DES");
            Cipher cipher=Cipher.getInstance("DES");
            cipher.init(Cipher.ENCRYPT_MODE, skypespec);
            byte[] encrypted=cipher.doFinal(userClearText.getBytes());
            userInfo=new String(encrypted);
        } catch (Exception e) {
            e.printStackTrace();
            throw new Exception(e);
        }
        return userInfo;
    }
}
```

Recommendation: Always use the strongest encryption algorithm

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

## Code Review for Use of Insufficient Encryption Key Size



Code Pattern: Look for code where the developer is attempting to set an insufficient encryption key size

```
File Edit Source Refactor Navigate Search Project Run Window Help
Save (Ctrl+S) loadedClass.java
10 public static String encrypt(String strClearText, String strKey) throws Exception{
11     String userInfor...
12
13     try {
14
15         KeyPairGenerator skeyspec = KeyPairGenerator.getInstance("RSA");
16         skeyspec.initialize(1024);
17         KeyPair kp = skeyspec.generateKeyPair();
18         PublicKey pubk = kp.getPublic();
19         PrivateKey privk = kp.getPrivate();
20
21
22     } catch (Exception e) {
23         e.printStackTrace();
24     }
25 }
```

Recommendation: Always use a secure algorithm with sufficient key size for setting passwords

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

## Code Review for Unsafe Decoding



Code Pattern: Look for code where the developer is attempting to decode cookie value, if cookies are not safely decoded, attacker can easily abuse the code

```
File Edit Source Refactor Navigate Search Project Run Window Help
Something.java
29 */
30 protected void doGet(HttpServletRequest request, HttpServletResponse response)
31
32     Cookie[] cookies=request.getCookies();
33     Cookie crosscookie=GetCookie(cookies,"test");
34     String FindCookieVal=crosscookie.getValue();
35     String decodecoookie=URLDecoder.decode(FindCookieVal, "UTF-8");
```

Recommendation: Always cross check the cookie value before use and confirm whether it is double encoded

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

## Code Review for Use of SSL



**Code Pattern:** Look for configuration where the developer is attempting to set a **false** value to the **SSLEnabled** attribute in **server.xml** file

 highlighted with a red box."/>

```
File Edit Source Navigate Search Project Run Window Help
Test.jsp *server.xml
76<
77<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="false"
78 maxThreads="150" scheme="https" secure="true"
79 clientAuth="false" sslProtocol="TLS" />
80
81<!-- Define an AJP 1.3 Connector on port 8009 -->
82<Connector port="8009" protocol="AJP/1.3" redirectPort="8443"/>
83
84
Design Source
server/Se.../#comment Writable Smart Insert 91:39
```

**Recommendation:** Always set the **SSLEnabled** attribute to **true**

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

## Code Review for OS Command Injection



**Code Pattern:** Look for code where the developer is allowing the **OS command** to execute from the application. Attackers can misuse this functionality and can execute any arbitrary command

```
File Edit Source Refactor Navigate Search Project Test Run Window Help
Save All (Ctrl+Shift+S) Login.jsp
12<
13<
14 String Hostname = request.getParameter("gethostname");
15<
16 Runtime runtime = Runtime.getRuntime();
17 runtime.exec("cmd.exe /C ping.exe" + Hostname);
18<
19<
20<
21<
22<
Design Preview
Writable Smart Insert 14:60
```

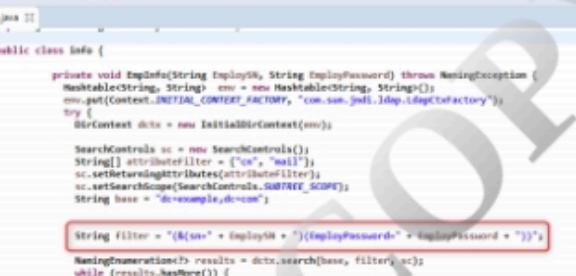
**Recommendation:** Developers should avoid executing OS command using **runtime.exec**. Instead use **Java API's** to execute any OS related command

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

## Code Review for LDAP Injection



■ **Code Pattern:** Look for code where the developer allows the search filter without validation so attackers can input any malicious character and invoke the method. Example: [EmpInfo\(\)](#)



The screenshot shows the Eclipse IDE interface with the Java editor open. The code is a Java class named 'Info' with a static method 'main'. The method performs an LDAP search using a JNDI context. It sets up a 'SearchControls' object with a filter '(cn=' + employId + ')', returning attributes 'cn' and 'mail'. The search scope is set to 'SUBTREE\_SCOPE'. The search base is 'dc=example,dc=com'. The code uses 'String' literals for the filter and base, which are highlighted with a red box.

```
private void EmpInfo(String EmployId, String EmployPassword) throws NamingException {
    Hashtable<String, String> env = new Hashtable<String, String>();
    env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.SECURITY_AUTHENTICATION, "simple");
    env.put(Context.SECURITY_PRINCIPAL, "uid=" + EmployId + ",ou=people,dc=example,dc=com");
    env.put(Context.SECURITY_CREDENTIALS, EmployPassword);
    LdapContext ctx = new InitialDirContext(env);

    SearchControls sc = new SearchControls();
    String[] attributeFilter = {"cn", "mail"};
    sc.setReturningAttributes(attributeFilter);
    sc.setSearchScope(SearchControls.SUBTREE_SCOPE);
    String base = "dc=example,dc=com";

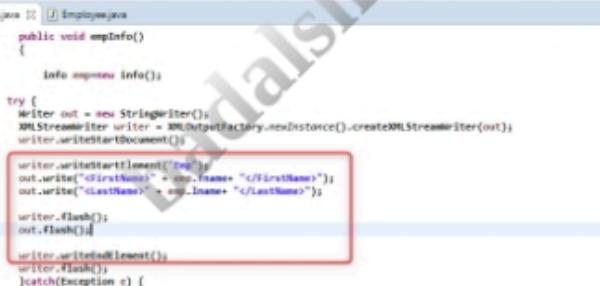
    String filter = "(cn=" + EmployId + ")(EmployPassword=" + EmployPassword + ")";
```

**Recommendation:** Always use a [whitelist](#) to validate user input so the user input contains only valid characters such as passwords containing only alphanumeric characters

## Code Review for XML Injection



- Code Pattern: Look for code where the developer is attempting to use xml for writing data streams `write()` method which doesn't apply any **meta character encoding**



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

- Menu Bar:** File, Source, Refactor, Navigate, Search, Object, Run, Window, Help.
- Toolbar:** Includes icons for New, Open, Save, Cut, Copy, Paste, Find, Select All, Undo, Redo, etc.
- Quick Access:** A search bar at the top right.
- Left Margin:** Shows line numbers (1-49) and a small icon (#).
- Code Editor:** Displays Java code for generating XML output. A red box highlights the section from line 33 to line 46, which writes elements for first name and last name.

```
public void empInfo()
{
    Info response info();

    try {
        Writer out = new StringWriter();
        XMLStreamWriter writer = XMLOutputFactory.newInstance().createXMLStreamWriter(out);
        writer.writeStartDocument();
        writer.writeStartElement("emp");
        out.write("firstName" + info.getName() + "</FirstName>");
        out.write("lastName" + info.getName() + "</LastName>");

        writer.flush();
        out.flush();
        writer.writeEndElement();
        writer.flush();
    } catch(exception e) {
    }
}
```

**Recommendation:** Always use **meta character encoding** while writing data into xml

## Code Review for Unsafe use of request parameter to execute SQL Query

CASE  
Classical Application Security Expertise

- Code Pattern: Look for code where the developer is attempting to execute a query directly received as **request parameter**. Attackers can manipulate these request parameters and execute a malicious query

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

- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Test, Run, Window, Help.
- Toolbar:** Includes icons for New, Open, Save, Cut, Copy, Paste, Find, Replace, Select All, Undo, Redo, etc.
- Quick Access:** A dropdown menu listing recent files and projects.
- Left Sidebar:** Shows the current file is "info.java" and the project is "Login.jsp".
- Code Editor:** Displays Java code for a JSP page. The code includes imports for java.util, javax.servlet, javax.servlet.http, and javax.servlet.jsp. It defines a class named "info" with a main method that prints "Hello World". Below this, there is code for a JSP page with a title, head, body, and scriptlet. The scriptlet contains JDBC code to connect to an Oracle database and execute a query. The line "ps=con.prepareStatement(sql);" is highlighted with a red box.

**Recommendation:** Never use a **request parameter** for executing a query

Downloaded by [LMU Muenchen] at 06:20 17 March 2015. Reproduced by permission.

## Code Review for Insecure File Upload

CASE  
Certified Application Security Expert

**Code Pattern:** Look for code where the developer is attempting to upload a file without proper validation. An attacker can send any file and execute a command from the file



The screenshot shows a Java IDE interface with a code editor containing Java code. The code is for handling file uploads, specifically reading a file part from a request, getting its name, and then writing its content to a file output stream. The code is annotated with several red boxes highlighting specific lines of code:

- A red box surrounds the line: `Part filePart = request.getPart("file");`
- A red box surrounds the line: `OutputStream out = new FileOutputStream(UploadPath + fileName);`

```
1 package com.dashsoft;
2
3 import java.io.*;
4 import javax.servlet.*;
5 import javax.servlet.http.*;
6
7 public class Info {
8
9     try{
10         Part filePart = request.getPart("file");
11         String fileName = filePart.getName();
12         boolean fileUploaded = false;
13
14         if(filePart != null && fileName != null){
15             InputStream fileContent = filePart.getInputStream();
16             byte[] bytes = IOUtils.toByteArray(fileContent);
17
18             OutputStream out = new FileOutputStream(UploadPath + fileName);
19             out.write(bytes);
20             fileUploaded = true;
21
22             fileContent.close();
23             out.flush();
24             out.close();
25         }
26     }
27 }
```

**Recommendation:** Always validate file extensions while uploading files

Downloaded by [128.111.121.42] at 06:45 04 September 2016

## Code Review for Directory Traversal



**Code Pattern:** Look for code where the developer is attempting to read a file if the file exists. Attackers can manipulate path variables and can access **restricted files and directories**

```
String fileName = request.getParameter("file");
if (fileName == null) {
    return;
}
File downloadedfile = new File(UploadPath + fileName);
if (!downloadedfile.exists()) {
    return;
}
OutputStream outi = null;
FileInputStream in = null;
try {
    outi = response.getOutputStream();
    in = new FileInputStream(downloadedfile);
    byte[] buffer = new byte[4096];
    int length;
    while ((length = in.read(buffer)) > 0) {
        outi.write(buffer, 0, length);
    }
}
```

**Recommendation:** Always validate the location of a file and read file from that location

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

## Code Review for Sensitive Information Exposure



**Code Pattern:** Look for code where the developer is attempting to store sensitive information into a **cookie**. A cookie can be read by attackers

```
import java.io.IOException;
public class ValidateUser {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        String currentDir = System.getProperty("user.dir");
        Cookie cookie=new Cookie("cud", currentDir);
        res.addCookie(cookie);
    }
}
```

**Recommendation:** Do not store sensitive information in cookies

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

## Code Review for Sensitive Information Leakage



**Code Pattern:** Look for code where the developer is using `printStackTrace` while handling exceptions. `printStackTrace` gives detailed information about the exception occurred

```
File Edit Source Refactor Navigate Search Project Text Run Window Help
ValidateUser.java Login.jsp
36     ResultSet rs=null;
37     int maxid=0;
38     try{
39         Class.forName("oracle.jdbc.driver.OracleDriver");
40         con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","system","admin");
41         ps=con.prepareStatement("select max(userid) from customerlist");
42
43         rs=ps.executeQuery();
44         while(rs.next()){
45             maxid=rs.getInt(1)+1;
46
47         }
48     }catch(Exception e){
49         e.printStackTrace();
50     }
Design Preview
Markers Properties Servers Data Source Explorer Snippets Console REST Annotations WebLogic Web Service Annotations
Writable Smart Insert 48 | 20
html/body/jspscriptlet#Text
```

**Recommendation:** Do not print the `stack trace` when the exception is caught

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

## Code Review for Generic Exception Throwing and Catching



**Code Pattern:** Look for code where the developer is attempting to `throw` and `catch` an exception with super class

```
File Edit Source Refactor Navigate Search Project Text Run Window Help
BackendLogic.jsp
104     byte[] bytes = md.digest();
105     StringBuilder sb = new StringBuilder();
106     for(int i=0; i< bytes.length ;i++) {
107         sb.append(Integer.toHexString((bytes[i] & 0xFF) + 0x100, 16).substring(1));
108     }
109     generatedPassword = sb.toString();
110     throw new Exception();
111 }
112 catch (NoSuchAlgorithmException e)
113 {
114     e.printStackTrace();
}
File Edit Source Refactor Navigate Search Project Text Run Window Help
ValidateUser.java
1    package com;
2
3    import java.sql.Connection;
4    import java.sql.PreparedStatement;
5    import java.sql.ResultSet;
6    import java.sql.Statement;
7
8    public class ValidateUser {
9        Connection con;
10       Statement ps;
11       ResultSet rs;
12
13       public void validateUser(String username, String password) {
14           try {
15               ps=con.prepareStatement("select max(userid) from customerlist");
16
17               rs=ps.executeQuery();
18               while(rs.next()){
19                   maxid=rs.getInt(1)+1;
20
21               }
22           }catch(Exception e){
23
24               e.printStackTrace();
25           }
26       }
27   }

```

**Recommendation:** Always `throw` and `catch` exceptions with a specific type of `exception class`

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

## Code Review for Cookies Vulnerable to Client-side Scripts Attacks



Code Pattern: Look for the configuration where the developer is attempting to set the **httpOnly** attribute to **false**

The screenshot shows two code editors side-by-side. The left editor displays a Java file named 'Test.jsp' containing JSP code. The right editor displays a Java file named 'CheckLogin.jsp' containing Java code. Both files are annotated with red boxes highlighting specific lines of code related to cookie configuration.

**Test.jsp:**

```
<xml version="1.0" encoding="UTF-8">
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
    <display-name>Mod4Lab1</display-name>
    <welcome-file-list>
        <welcome-file>Home.jsp</welcome-file>
    </welcome-file-list>
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
    <cookie-config>
        <http-only>false</http-only>
    </cookie-config>
</web-app>
```

**CheckLogin.jsp:**

```
try {
    final String userID = "user";
    final String password = "pass";
    String user = request.getParameter("uname");
    String pwd = request.getParameter("pwd");
    if (userID.equals(user) && password.equals(pwd)) {
        Cookie loginCookie = new Cookie("user", user);
        loginCookie.setMaxAge(30 * 60);
        loginCookie.setSecure(true);
        loginCookie.setHttpOnly(false);
        response.addCookie(loginCookie);
        response.sendRedirect("index.jsp");
    } else {
        response.sendRedirect("Login.jsp");
    }
}
```

Recommendation: Always set **httpOnly** attribute to **true**

## Code Review for Cookies Vulnerable to CSRF Attacks



Code Pattern: Look for configuration where the developer is attempting to set the **session-timeout** value to a large value in the **web.xml** file

The screenshot shows a code editor displaying a Java file named 'Test.jsp'. The XML configuration section of the file is highlighted with a red box, indicating it as a point of interest for review.

**Test.jsp:**

```
<xml version="1.0" encoding="UTF-8">
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
    <display-name>Mod4Lab1</display-name>
    <welcome-file-list>
        <welcome-file>Home.jsp</welcome-file>
    </welcome-file-list>
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
</web-app>
```

Recommendation: Always set **session-timeout** value to as minimum as possible

## Code Review for Enabling Directory Listing



**Code Pattern:** Look for configuration where the developer is attempting to set the directory **listing parameter** value to **true** in web.xml file of the tomcat server

```
File Edit Source Navigate Search Project Run Window Help
BackEndReg.jsp server.xml *Java Secure Deployment Practices.pdf "web.xml" 12
94<init-param>
95    <param-name>debug</param-name>
96    <param-value>@</param-value>
97</init-param>
98<init-param>
99    <param-name>listings</param-name>
100   <param-value>true</param-value>
101</init-param>
102<load-on-startup>1</load-on-startup>
103</servlet>
104
```

**Recommendation:** Do not enable the directory listing feature. Set **param** value as "**false**" in web.xml

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

## Code Review: Check List Approach



What does a **Checklist** Contain?



Data Validation and Encoding Controls

Encryption Controls

Authentication and Authorization Controls

Session Management

Exception Handling

Auditing and Logging

Security Configurations

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

## Sample Checklist: Input Validation



- ✓ Ensure that all external inputs are validated
- ✓ Ensure that data is validated server-side (security should not rely on client-side validations)
- ✓ Ensure that the application validates respective data types and rejects unexpected input
- ✓ Ensure that the application efficiently evaluates input length
- ✓ Ensure that there is strong separation between data and commands
- ✓ Ensure that there is strong separation between data and client side scripts
- ✓ Ensure that data is validated for special characters before being passed to SQL, LDAP, OS and third party commands
- ✓ Ensure that http headers are validated for each request (e.g. referer)

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

## Sample Checklist: Authentication



- Ensure that user credentials are encrypted in the data store
- Ensure that security policies are configurable (not hardcoded)
- Ensure that standard security frameworks are used (instead of custom code)
- Ensure that SSL is used to protect user credentials and authentication tokens
- Ensure that authentication cookies are not persisted
- Ensure that authentication cookies are encrypted
- Ensure that cookie names and paths are used
- Ensure that application handles user management events such as authentication failure, password reset, password change, account lockout and cancel account
- Ensure that application handles suspicious events such as multiple failed logon attempts, session replay and attempted access to restricted resources
- Ensure that strong passwords policies are enforced
- Ensure that authentication credentials are not passed by HTTP GET

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

## Sample Checklist: Authorization



Ensure that authentication and authorization should be the first logic executed for each request

Ensure that authorization checks are granular (page and directory level)

Ensure that application denies access to pages and data by default

Ensure that application re-authenticates for requests that have side-effects

Ensure that ACLs are configured for all files

Ensure that authorization is performed based on roles defined

Ensure that application implements strong authorization and can not be bypassed through parameter manipulation

Ensure that authorization cannot be bypassed through cookie manipulation

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

## Sample Checklist: Session Management



Ensure that application should not pass session parameters in URLs

Ensure that duration for session cookies expiration should be as minimum as possible

Ensure that cookies are encrypted

Ensure that session data is validated

Ensure that private data in cookies is kept to a minimum

Ensure that application avoids excessive cookie use

Ensure that session id is complex and generated randomly

Ensure that session storage is secure

Ensure that application properly handles invalid session ids

Ensure that session timeout are enforced

Ensure that logout invalids the session

Ensure that session resources are released when session invalidated

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

## Sample Checklist: Cryptography



- Ensure that sensitive data is secured in memory, storage and transit
- Ensure that sensitive information is not passed to/from non-SSL pages
- Ensure that proper SSL is set up
- Ensure that SSL provider supports only strong algorithms
- Ensure that decryption services are protected by authentication/authorization
- Ensure that application securely stores cryptographic keys

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

## Sample Checklist: Exception Handling



- Ensure that the application fails securely
- Ensure that error messages do not reveal sensitive information
- Ensure that system errors are never shown to users
- Ensure that resources are released and transactions are rolled back when there is an error

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

## Sample Checklist: Logging



- Ensure that all user / system actions are logged
- Ensure that sensitive information is not logged (e.g. passwords)
- Ensure that Logging is enabled for user management events (e.g. password reset)
- Ensure that unusual activity such as multiple login attempts are logged
- Ensure that logs have enough detail to reconstruct events for audit purposes
- Ensure that logging is highly configurable (logging levels)

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

## SAST Finding



- |   |   |
|---|---|
| <ul style="list-style-type: none"><li>■ The SAST findings keep the development team informed about the findings gathered by the security team during the secure code review</li><li>■ The findings generally include vulnerabilities, architectural problems, organization issues and any mismatch in the best practices followed</li><li>■ The security team members may include the following features in the findings report:<ul style="list-style-type: none"><li>● <b>Select a meaningful title for the findings report:</b> The title should be clear and should convey the details of the issue clearly to the development</li><li>● <b>Determine clearly the location of the issue:</b> The development team should easily identify the location of the issue from the report</li><li>● <b>Provide detailed description of the security issues:</b> The security team should include the following details regarding the security issue:<ul style="list-style-type: none"><li>● A clear description about the vulnerability</li><li>● Analysis of the possible attack areas</li><li>● Description about the factors leading to the attack</li></ul></li></ul></li></ul> | <ul style="list-style-type: none"><li>■ <b>Provide a description about the risk ratings:</b> The risk rating enables the development team regarding the value assigned to each security issue found. The report may also contain the explanation behind assigning any particular value to an identified security issue that may enable the development team to understand the impact of the security issue at a glance. The commonly used risk ratings during code review are as follows:<ul style="list-style-type: none"><li>● Critical</li><li>● High</li><li>● Moderate</li><li>● Low</li></ul></li><li>■ The security team should follow some standards while assigning risk ratings to the security issues during secure code review</li><li>■ <b>Provide proper corrective measures to the development team:</b> The security team after the secure code review may provide certain corrective measures that enable the development team to easily rectify the security issues. The rectification measures may also include:<ul style="list-style-type: none"><li>● Other course of action</li><li>● Total amount of effort required for rectification</li></ul></li></ul> |
|---|---|

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

## SAST Report



■ The SAST report consists of the following details:

1 A list of vulnerabilities and its related details along with the severity level

2 The rectification measures taken for controlling the vulnerabilities and errors found in the code

3 Graphical representation of the security measures required for the application

4 Control measures that can be taken for future security as well

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

## SAST Reporting



- Report the findings of the code review in a structured way
- The report should include:
  - Weakness Metadata
    - Details of Place to find the issue
  - Through Description
    - Detailed description of the issue
  - Recommendation
    - Recommendation to mitigate the issue
  - Assign appropriate Priority
    - Give priority to the issue for the developer to prepare his to -do-list

### SAMPLE REPORT

#### SQL INJECTION

Location: \myProject\ UserManager\ User.jsp?username=

Description: The code below builds a dynamic SQL statement without validating the input from request parameter. This may lead to SQL INJECTION

```
String query="select FirstName,LastName from Users
WHERE UserName=" +request.getParameter("username")
+ """; PreparedStatement
stmt=con.prepareStatement(query);
ResultSet rs=stmt.executeQuery();
```

Recommendation: Use parameterized SQL query; do not dynamically concatenate the request parameter value to the string query.

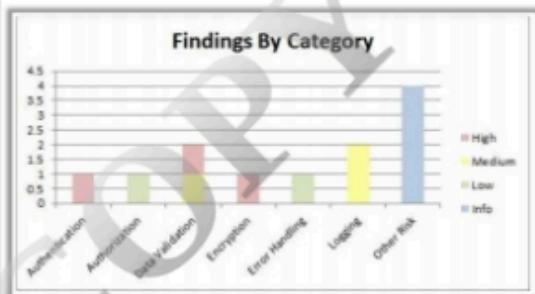
Ref:[https://www.owasp.org/index.php/SQL\\_Injection\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet)

## SAST Reporting (Cont'd)



Review /Engagement Reference:			
Package/Component/Class Name/Line Number:			
Finding Title:			
Severity: High			
Finding Description	Location(s)	Risk Description	Recommendation
No input validation of the <code>HttpServletRequest objects.getParameter()</code> function.	com.inc.dostuff.java Lines 20, 55,106	Description: of the likelihood and impact to the business if the flaw were to be exploited.	It is critical that this be addressed prior to deployment to production
Lack of input validation may make the application vulnerable to many types of injection	com.inc.main.java Lines 34, 99		

Finding Report Sample



Risk Rating

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



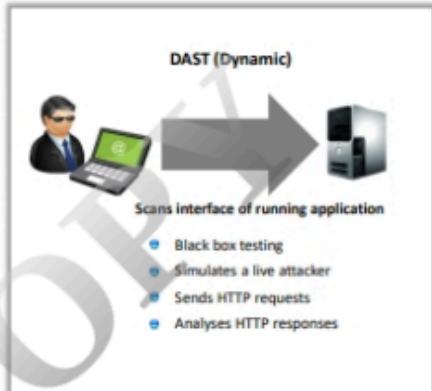
## Dynamic Application Security Testing

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

## Dynamic Application Security Testing (DAST)



- DAST is a security testing technique which involves **simulating** attacks against the application and analyzes how the application behaves
- The application is tested **dynamically** from the outside when application is running
- DAST is generally performed by the penetration testers or security practitioners on a working system in pre-production, a test environment, or even in production
- These individuals typically uses automated web application vulnerability scanners to conduct DAST
- These scanners believe in **fuzzing** the application inputs with attack payloads in order to detect security weaknesses
- Typical web application vulnerability scanners are classified into:
  - **Classic application security scanners:** They crawl and automatically scan all detected webpages. However, they have limited coverage within the application as they do not have understanding of business flows
  - **Proxy-based security testing tools:** These are scanners integrated in web application proxies which provide fine-grained control over each request that gets scanned



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

## Types of DAST



### Types of DAST

- Automated Application Vulnerability Scanning
  - Security tester uses classic **application security scanners** to scan the web application for vulnerabilities

- Manual Application Security Testing
  - Security tester uses **Proxy-based security testing tools** to craft and send request manually and analyze the responses from the application

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

## SAST vs DAST



SAST	DAST
White box security testing	Black box security testing
Requires a source code	Requires a running application
Finds vulnerability earlier in SDLC	Finds the vulnerability towards the end of SDLC
Less expensive to fix vulnerability	More expensive to fix vulnerability
Runtime and environment related issues can't be discovered	Runtime and environment related issues can be discovered
Typically supports all kinds of software	Typically scans only apps like web application and web services

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



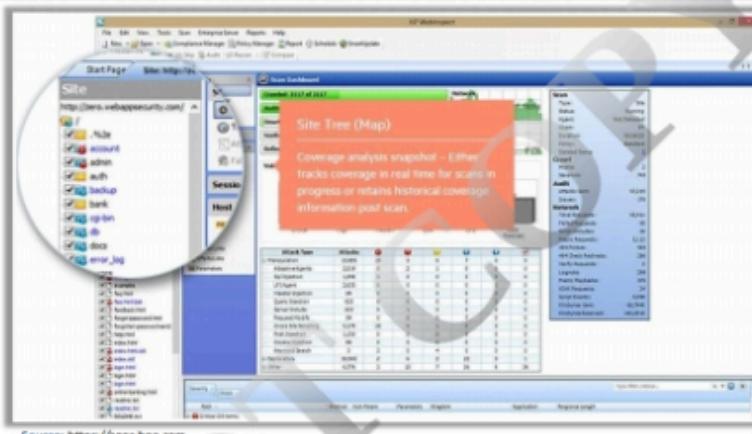
## Automated Application Vulnerability Scanning Tools

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

## Web Application Security Scanners: WebInspect



- WebInspect is a web application security assessment solution designed to thoroughly analyze today's complex **web applications** and **web services** for security vulnerabilities
- It provides the broadest dynamic application security testing (DAST) coverage and detects new types of vulnerabilities that often go undetected by **black-box security testing** technologies



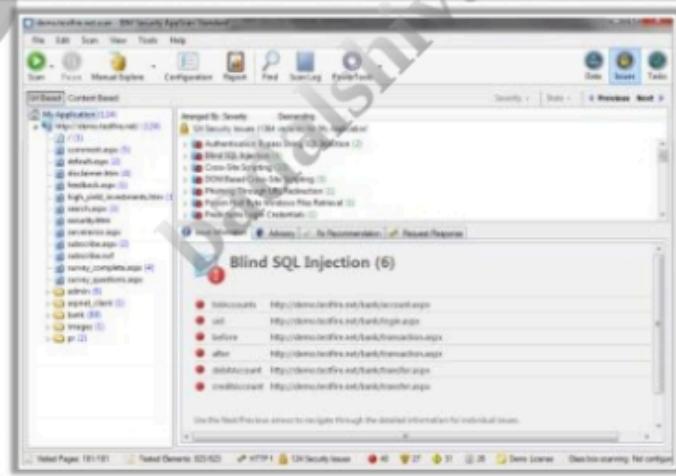
Source: <https://soas.hpe.com>

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

## Web Application Security Scanners: IBM Security AppScan



- IBM Security AppScan helps organizations decrease the likelihood of web application attacks and costly data breaches by automating application security vulnerability testing



Source: <https://www.ibm.com>

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

## Additional Web Application Vulnerability Scanners



**Acunetix WVS**  
<https://www.acunetix.com>



**Websecurity**  
<https://www.websecurity.com>



**Nessus**  
<https://www.tenable.com>



**NeXpose**  
<https://www.rapid7.com>



**ParosPro**  
<https://www.milescon.com>



**AppScan**  
<https://www.ibm.com>



**N-Stalker**  
<https://www.nstalker.com>



**NetSparker**  
<https://www.netsparker.com>



**WebKing**  
<https://www.parosoft.com>



**Retina Web Security Scanner**  
<https://www.beyondtrust.com>

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



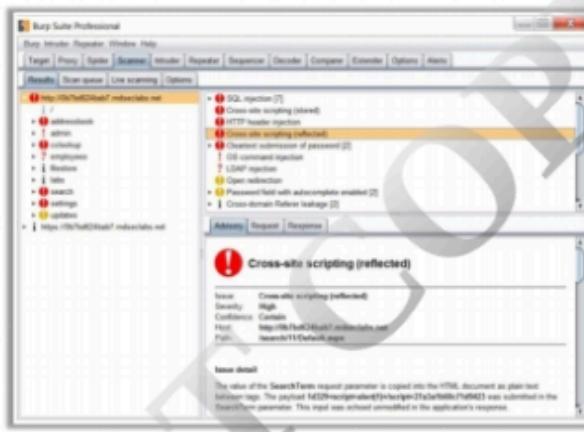
## Proxy-based Security Testing Tools

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

## Burp Suite



- Burp Suite is an integrated platform for performing security testing of web applications. Its various tools work seamlessly together to support the entire testing process, from initial mapping and analysis of an application's attack surface, through to finding and exploiting security vulnerabilities
- Install and configure Burp Suite in the browser of your interest



Source: <https://portswigger.net>

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

## OWASP Zed Attack Proxy (ZAP)



- OWASP ZAP is an open source, easy to use, integrated penetration testing tool for finding vulnerabilities in web applications

### DAST using OWASP ZAP

1. Install and configure ZAP in the browser of your interest
2. Insert the URL of web application that you want to test and start the attack



Source: <https://www.owasp.org>

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

## Additional Proxy-based Security Testing Tools



 **Vega**  
<https://www.acunetix.com>

 **Paros proxy**  
<http://www.parosproxy.org>

 **Firefox Tamper Data**  
<https://addons.mozilla.org>

 **WebScarab**  
<https://www.owasp.org>

 **Wfuzz**  
<http://www.edge-security.com>

 **Ratproxy**  
<https://code.google.com>

 **Webstretch**  
<https://sourceforge.net>

 **Wapiti**  
<http://wapiti.sourceforge.net>

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

## Choosing Between SAST and DAST



- 1** **SAST** is **white box** security testing approach while **DAST** is a **black box** security testing approach
- 2** These approaches work differently on their part to **find security flaws** in the application
- 3** **Relying on DAST** alone does not ensure that security flaws will **exist** that you have not tested for
- 4** Relying on SAST alone does not ensure that security flaw will not exist that may arise when application is running
- 5** After SAST, security tester may have a better chance of **finding different problems** when application is running
- 6** Thus, both **SAST** and **DAST** are required in a secure software development program

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

## Module Summary



- SAST is one of the software security assurance approaches executed in an attempt to identify security-related weaknesses in the code
- Manual Source Code Review involves manually inspecting the source code line by line to detect any defects and security related flaws
- The static code analysis tools automate the entire process of secure code review; however, they scan the code superficially
- With manual secure code review, you can analyze and detect many security vulnerabilities especially those missed out during the static code analysis
- DAST is a security testing technique which involves simulating attacks against the application and analyzing how the application behaves
- SAST is a white box security testing approach while DAST is a black box security testing approach

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

DO NOT COPY  
This page is intentionally left blank.  
[badalshiva@gmail.com](mailto:badalshiva@gmail.com)



## Module 10

# Secure Deployment and Maintenance

DO NOT COPY  
This page is intentionally left blank.  
[badalshiva@gmail.com](mailto:badalshiva@gmail.com)

## Module Objectives



- 1 Understand the Importance of Secure Deployment
- 2 Discuss Security Practices at Host Level
- 3 Discuss Security Practices at Network Level
- 4 Discuss Security Practices at Application Level
- 5 Discuss Security Practices at Web Container Level (Tomcat)
- 6 Discuss Security Practices at Oracle Database Level
- 7 Discuss Security Maintenance and Monitoring Activities

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

## Secure Deployment



- 1 Deployment is the **last phase** of software development lifecycle where the application is moved from development environment to production environment
- 2 It is the first time where the **system administrator** is involved in software development
- 3 The application can be exposed to different types of attacks, if an administrator commits **configuration mistakes** in the deployment phase
- 4 **Security** should also be the key consideration while deploying any application

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

## Prior Deployment Activity: Check the Integrity of Application Package before Deployment



- Software vendors generally provide **checksum** of the application with the application package
- System administrators should again generate a checksum of the application and compare it with the checksum provided with the package to ensure the integrity of the application
- Two **identical checksums** confirm that the application is not tampered during the delivery

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

## Prior Deployment Activity: Review the Deployment Guide Provided by the Software Vendor



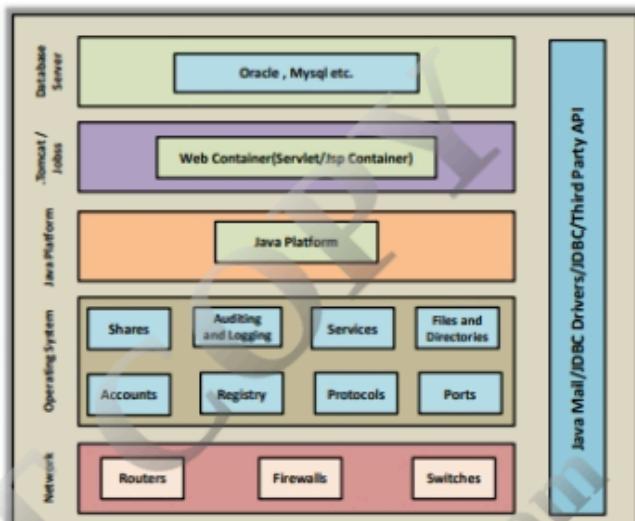
- Use **deployment checklists** provided by the software vendor
- **Do not build and use** your own deployment checklist as it may affect the application security

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

## Deployment Activities: Ensuring Security at Various Levels



- ❑ Secure deployment involves ensuring security at various levels from bottom to top
- ❑ It involves considering security related to the potential customer environments which include:
  - ❑ Host Level Deployment Security
  - ❑ Container Level (Tomcat) Deployment Security
  - ❑ Oracle Level Deployment Security



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



### Ensuring Security at Host Level

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

## Check and Configure the Security of Machine Hosting Web Server, Application Server, Database Server and Network Devices



I

Firstly, the production environment will be secured, if the machine hosting the **web server** is secured

II

The physical security of a host machine, its OS security, and the security of all other software installed on the machine should be ensured

III

Check the **host server** and all of the services upon which the application security will depend

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

## Physical Security



Ensure that **server machine is placed** in a secured area

Ensure that an appropriate physical access control is implemented to restrict unauthorized physical access



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

## Host Level Security



Component	Characteristics
Patches and Updates	<ul style="list-style-type: none"><li>⊕ Not deploying patches and updates to the web server can allow attackers to run malicious code or intrude the server</li></ul>
Services	<ul style="list-style-type: none"><li>⊕ Services are easy targets for attackers to access the privilege and exploit the functioning of the server</li><li>⊕ Do not run the service that is not required for the server functioning</li><li>⊕ It is important to monitor running services</li></ul>
Protocols	<ul style="list-style-type: none"><li>⊕ Do not implement protocols that are insecure and can not provide encrypted authentication</li><li>⊕ Example: Telnet, File Transfer Protocol(FTP), Simple Mail transfer Protocol(SMTP)</li></ul>
Accounts	<ul style="list-style-type: none"><li>⊕ Audit the user accounts regularly to avoid any malicious activities</li><li>⊕ Provide limited privileges to account to prevent privilege elevation</li><li>⊕ Delete accounts that are no more beneficial for the application</li><li>⊕ Implement strong password policies</li></ul>
Files and Directories	<ul style="list-style-type: none"><li>⊕ Implement NTFS permissions on files and directories on web server this will allow access to only Windows services</li></ul>
Shares	<ul style="list-style-type: none"><li>⊕ Secure file shares process with NTFS permissions</li><li>⊕ Limited shares reduces intrusion possibility on web server</li></ul>
Ports	<ul style="list-style-type: none"><li>⊕ Services running on server listen to specific ports</li><li>⊕ Auditing ports regularly will help to detect any irregular service on the server</li></ul>
Registry	<ul style="list-style-type: none"><li>⊕ Secure the registry by:<ul style="list-style-type: none"><li>⊖ Applying restricted Windows ACL</li><li>⊖ Blocking remote registry administration</li></ul></li></ul>

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



## Ensuring Security at Network Level

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

## Network Level Security: Router



Component	Characteristics
Patches and Updates	<ul style="list-style-type: none"> <li>Router operating system is patched with up-to-date software</li> <li>Unused protocols and ports are blocked</li> <li>Ingress and egress filtering is implemented</li> <li>ICMP traffic is screened from the internal network</li> </ul>
Protocols	<ul style="list-style-type: none"> <li>TTL expired messages with values of 1 or 0 are blocked (route tracing is disabled)</li> <li>Directed broadcast traffic is not forwarded</li> <li>Large ping packets are screened</li> <li>Routing Information Protocol (RIP) packets, if used, are blocked at the outermost router</li> </ul>
Administrative access	<ul style="list-style-type: none"> <li>Unused management interfaces on the router are disabled</li> <li>A strong administration password policy is enforced</li> <li>Static routing is used</li> <li>Web-facing administration is disabled</li> </ul>
Services	<ul style="list-style-type: none"> <li>Unused services are disabled (for example bootps and finger)</li> </ul>
Auditing and logging	<ul style="list-style-type: none"> <li>Logging is enabled for all denied traffic</li> <li>Logs are centrally stored and secured</li> <li>Auditing against the logs for unusual patterns is in place</li> </ul>
Intrusion detection	<ul style="list-style-type: none"> <li>IDS is in place to identify and notify an active attack</li> </ul>

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

## Network Level Security: Firewall



Component	Characteristics
Patches and Updates	<ul style="list-style-type: none"> <li>Firewall software and OS are patched with latest security updates</li> </ul>
Filters	<ul style="list-style-type: none"> <li>Packet filtering policy blocks all but required traffic in both directions</li> <li>Application-specific filters are in place to restrict unnecessary traffic</li> </ul>
Logging and Auditing	<ul style="list-style-type: none"> <li>All permitted traffic is logged</li> <li>Denied traffic is logged</li> <li>Logs are cycled with a frequency that allows quick data analysis</li> <li>All devices on the network are synchronized to a common time source</li> </ul>
Perimeter networks	<ul style="list-style-type: none"> <li>Perimeter network is in place if multiple networks require access to the server</li> <li>Firewall is placed between untrusted networks</li> </ul>

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

## Network Level Security: Switch



Component	Characteristics
Patches and Updates	<ul style="list-style-type: none"><li>⊕ Latest security patches are tested and installed, or the threat from known vulnerabilities is mitigated</li></ul>
VLANs	<ul style="list-style-type: none"><li>⊕ Make sure VLANs are not overused or overly trusted</li></ul>
Insecure defaults	<ul style="list-style-type: none"><li>⊕ All factory passwords are changed</li><li>⊕ Minimal administrative interfaces are available</li><li>⊕ Access controls are configured to secure SNMP community strings</li></ul>
Services	<ul style="list-style-type: none"><li>⊕ Unused services are disabled</li></ul>
Encryption	<ul style="list-style-type: none"><li>⊕ Switched traffic is encrypted</li></ul>

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



## Ensuring Security at Application Level

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

## Web Application Firewall (WAF)



1

A traditional firewall cannot secure web servers from a malicious traffic attack as the attack occurs at layer 7 of the network stack

2

A **WAF** provides a security layer that protects the **web server** from the malicious traffic

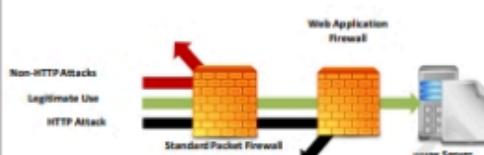
3

WAF is either **appliance based** or **cloud based** and gets deployed through a proxy placed ahead of the web application

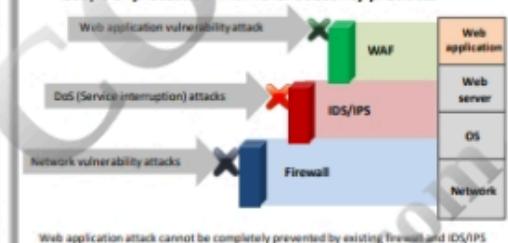
4

It uses a **rule based filter** that monitors and analyzes the traffic before it reaches the web application

### Placement of WAF and its Working



### Scope of protection in different security products



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

## Benefits of WAF



WAF implementation secures the **existing and productive web applications**

Many WAF's have functionalities that can be used in design process and minimizing the work load

WAF provides **cookies protection** with encryption and signature methodology

It secures applications from **cross-site request forgery** and negates **parameter tampering** by **URL encryption**

WAF can detect **data validation issues** by in-depth testing of character, character length, the range of a value, etc.

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

## WAF Limitations



- Do not consider WAF as a replacement for a proper **application security** such as user authentication or input filtering
- WAF is not a technology which if once deployed, can be ignored by the administrator
- The working of WAF is different from the **next-generation firewall (NGFW)**. WAF inspects traffic based on a particular protocol unlike NGFW which can make changes in the existing network
- WAF does not provide complete security from all web attacks as it can not read database commands
- Only if WAF manages the session itself, it can partially prevent issues like session fixation and anti-automation
- Deployment of WAF does not ensure security from **false positives**

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

## WAF Vendors



**Barracuda WAF**

Source: <http://www.barracuda.com>



**A10 WAF**

Source: <https://www.a10networks.com>



**Citrix NetScaler MPX WAF**

Source: <https://www.citrix.co.in>



**Imperva WAF**

Source: <https://www.imperva.com>



**MonitorApp WAF**

Source: <http://www.monitorapp.com>



**Trustwave WAF**

Source: <https://www.trustwave.com>

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

**Ensuring Security at Web Container Level (Tomcat)**

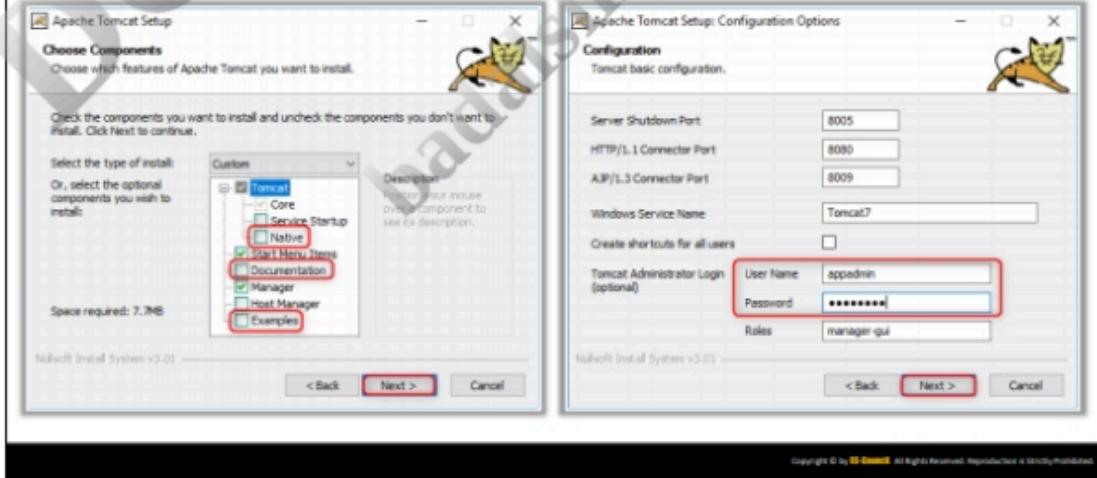
Copyright © by EC-Council® All Rights Reserved. Reproduction is Strictly Prohibited.



This slide discusses ensuring security at the web container level using Tomcat. It features a large watermark reading "DRAFT COPY" diagonally across the page. The top right corner contains the CASE logo with "JAVA" written vertically above it. The title "Ensuring Security at Web Container Level (Tomcat)" is centered at the top. A copyright notice at the bottom states "Copyright © by EC-Council® All Rights Reserved. Reproduction is Strictly Prohibited."

**Install and Configure Tomcat Securely**

While installing Tomcat uncheck the options Native, Documentation, and Examples in Choose Components window  
Select User Name other than admin, and give a secure Password as per the organization password policy in Configuration window



The slide shows two screenshots of the Apache Tomcat Setup interface. The left screenshot, titled "Apache Tomcat Setup: Choose Components", shows the "Custom" component selection screen. The "Native" checkbox is unchecked, while "Documentation" and "Examples" are checked. The right screenshot, titled "Apache Tomcat Setup: Configuration Options", shows the configuration settings. The "User Name" field is set to "appadmin" and the "Password" field is set to a masked value. Both fields are highlighted with a red border. Other configuration options like "Server Shutdown Port" (8005), "HTTP/1.1 Connector Port" (8080), and "Windows Service Name" (Tomcat7) are also visible.

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

## Remove Server Banner



- The **server banner** gives out the product and version details which leads to information disclosure vulnerability
  - To prevent Tomcat from displaying product and version details, remove the banner from the **HTTP header**
  - To **remove the server banner**
    - Open \$Tomcat/conf folder
    - Edit server.xml file
    - Set server="" in the Connector port
- Example: <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" Server="" redirectPort="8443" />

### Header Response with Server Banner

▼ General  
Request URL: http://192.168.100.123:8080/  
Request Method: GET  
Status Code: 200 OK  
Remote Address: 192.168.100.123:8080  
Referrer Policy: no-referrer-when-downgrade

▼ Response Headers view source  
Content-Type: text/html;charset=ISO-8859-1  
Date: Mon, 22 Jan 2018 08:23:06 GMT  
Server: Apache-Coyote/1.1  
Transfer-Encoding: chunked

### Header Response without Server Banner

▼ General  
Request URL: http://192.168.100.123:8080/  
Request Method: GET  
Status Code: 200 OK  
Remote Address: 192.168.100.123:8080  
Referrer Policy: no-referrer-when-downgrade

▼ Response Headers  
Content-Type: text/html;charset=ISO-8859-1  
Date: Mon, 22 Jan 2018 08:31:44 GMT  
Server:  
Transfer-Encoding: chunked

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

## Start Tomcat with Security Manager



- Security manager prevents running **untrusted applets**
- It protects browser from executing **servlets, JSPs, JSP beans, and tag libraries** containing trojan
- To run Tomcat with security manager
  - Add the security manager entries to the JAVA\_OPTS environment variable
  - Open catalina.policy file located at <Tomcat\_Home>/conf/ in a text editor
  - Alter the SecurityManager permission
  - Start Tomcat with security parameter

### Starting Tomcat with Security Manager

```
$CATALINA_HOME/bin/catalina.sh start -security [Unix]  
%CATALINA_HOME%\bin\catalina start -security [Windows]
```

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

## Configure Default Servlet Not to Serve Index Pages



- Ensure that the default servlet configuration settings do not render index pages in the absence of welcome files
- Edit the **listings** to **false** in **CATALINA\_HOME/conf/web.xml** file to configure the server not to serve index pages in absence of welcome files

### Default Servlet Configuration to not Serve Index Pages in Absence of Welcome Files

```
File Edit Source Navigate Search Project Run Window Help
[E] web.xml [Source]
97 <servlet>
98   <servlet-name>default</servlet-name>
99   <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
100  <init-param>
101    <param-name>debug</param-name>
102    <param-value>0</param-value>
103  </init-param>
104  <init-param>
105    <param-name>listings</param-name>
106    <param-value>false</param-value>
107  </init-param>
108  <load-on-startup>1</load-on-startup>
109 </servlet>
110
Design Source
```

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

## Replace Default Error Page



- 1 Default error page displays the entire **StackTrace** resulting in sensitive information disclosure
- 2 Edit **CATALINA\_HOME/conf/web.xml** file to replace the default error. This will display a blank page and protects the application from displaying the StackTrace to the user
- 3 Configure an error page in **CATALINA\_HOME/webapps/APP\_NAME/WEB-INF/web.xml** to display a custom error page

### Configuration to Replace Default Error Page

```
File Edit Source Navigate Search Project Run Window Help
[E] web.xml [Source]
18 </session-config>
19 <error-page>
20   <exception-type>java.lang.Throwable</exception-type>
21   <location>/error.jsp</location>
22 </error-page>
23 </web-app>
```

### Configuring Custom Error Page in web.xml

```
File Edit Source Navigate Search Project Run Window Help
[E] web.xml [Source]
19 <error-page>
20   <error-code>500</error-code>
21   <location>/ErrorPages/error.jsp</location>
22 </error-page>
23 <error-page>
24   <exception-type>java.lang.Throwable</exception-type>
25   <location>/ErrorPages/error.jsp</location>
26 </error-page>
27 </web-app>
```

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

## Replace Default server.xml



- Replace the **server.xml** with **server-minimal.xml** which contains the minimum required configuration

### To replace server.xml

- Rename **server.xml** file present in CATALINA\_HOME/conf/ to /originalserver.xml

- Rename **server-minimal.xml** file present in CATALINA\_HOME/conf/ to server.xml

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

## Protect Shutdown Port



- Tomcat listens to **8005 port** (default) for **shutdown commands**
- When running a publicly accessible server, it should be ensured that external access to the shutdown port is prevented by using a firewall
- To prevent unintended shutting down of Tomcat services, disable the shutdown port
- To disable the shutdown port
  - Edit the port attribute to **-1** in the **server.xml** file
- Setting **port attribute** to **-1** ensures that Tomcat is shutdown only by the user who owns the Tomcat process
- Alter the shutdown command by changing the **shutdownphrase** in the **CATALINA\_HOME/conf/server.xml** file
- Ensure that Tomcat users only have read rights for **server.xml** file

### Disabling Shutdown Port

```
web.xml  server.xml
19 DOCUMENTATION_AT /DOCS/CONT18/SERVER.HTML
20 --><Server port="-1" shutdown="SHUTDOWN">
```

```
web.xml  server.xml
19 DOCUMENTATION_AT /DOCS/CONT18/SERVER.HTML
20 --><Server port="8005" shutdown="ComplexPassword">
```

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

## Restrict Access to Tomcat Manager Applications



Make sure to restrict the IP addresses that can access the Tomcat Manager or administrative web applications

To enable connections only from the localhost (127.0.0.1) on the Tomcat server

1

Edit context element of \${tomcat\_home}/conf/Catalina/localhost/admin.xml

2

Edit context element of \${tomcat\_home}/conf/Catalina/localhost/manager.xml

```
25 <!-->
26<Context antiResourceLocking="false" privileged="true">
27 <Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="127.0.0.1"/>
28 </Context>
```

To enable connections using host name use RemoteHostValve

```
40 <Context>
41 <Valve className="org.apache.catalina.valves.RemoteHostValve" allow="*.ucarl.edu" />
42 </Context>
```



## Protecting Resources with Realms



Realm is a database of usernames and passwords for authentication of web applications



server.xml file has many realms



Make sure to enable useful realms and others should be disabled



LockOutRealm prevents brute force attacks and denial of service attacks by locking users, with the number of failed login attempts

Example: server.xml to implement lock out functionality to a UserDatabase Realm

```
160 <Realm className="org.apache.catalina.realm.LockOutRealm" >
161 <Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName="UserDatabase" />
162 </Realm>
```