

# Group 06

## Assignment 4

Final Report

*Complementary Hand in*

2020-12-03

Ranjana Ghimire  
Justas Narusas  
Efsthios Psyllakis  
Oskar Elfving Söderström  
Motti Aimé

# Table of Contents

<b>Time Allocation</b>	6
<b>Jforum Architecture</b>	8
3. Vulnerabilities	11
Improper Input Validation	11
Consequences	13
Resolution:	13
Consequences:	13
Solution:	14
Consequences	16
Resolution	16
Dependencies	16
Consequences	17
Resolution:	17
SQL Injection	17
Consequence	19
Resolution	19
OS Injection	19
Sensitive Data Protection	20
Consequence	21
Resolution	21
Credential Management in Broken Authentication	21
Resolution	22
Insufficient Access Control Mechanism	22
Consequence	23
Resolution	23
Malicious and incorrect HTML tags	23

Consequences	24
Resolution	24
Handling special characters	24
Consequences	24
Resolution	25
XSS Cross-Site Scripting	25
Consequence	26
Resolution	26
Cookies and Session Hijacking	26
Consequence	27
Resolution	27
<b>4. Static Tool analysis</b>	29
Coverity	29
Fortify	30
OWASP Dependency checker	32
References	34
Appendix	38

## Table of Figures

Figure 1 jForum Architecture .....	8
Figure 2 Code section which lacks input validation .....	11
Figure 3 Error message from invalid tags in email address.....	12
Figure 4 Source code inspection in web page.....	12
Figure 5 Code Section for Biography request defination .....	14
Figure 6 Insertion of HTML tags in Biography.....	15
Figure 7 Biography result after adding html tags .....	15
Figure 8 8 Dependency check via OWASP dependency check software.....	16
Figure 9 Code section lacking prepared statement .....	18
Figure 10 Result after SQL Injection in email address for forget password .....	18
Figure 11 code section lacking validation .....	18
Figure 12 Result after update tries in forum text .....	19
Figure 13 Output shows username without login .....	20
Figure 14 md5 hashes of passwords in database .....	22
Figure 15 Admin Panel with role listing.....	23
Figure 16 Adding javascript in biography .....	25
Figure 17 Result after adding Javascript in Biography.....	25
Figure 18 Cookies showing userid.....	26
Figure 19 Cookies defination code with userid defination .....	27
Figure 20 Result from Coverity .....	29
Figure 21 Script executed to run code analysis in all folders .....	30
Figure 22 Result from Coverity .....	32
Figure 23 Highly critical result from Fortify .....	32
Figure 24 Execution of OWASP dependency checker .....	33
Figure 25 Result from OWASP dependency checker.....	33

## Table of Tables

Table 1 High to Critical vulnerabilities detected by Fortify .....	30
--	----



## Time Allocation

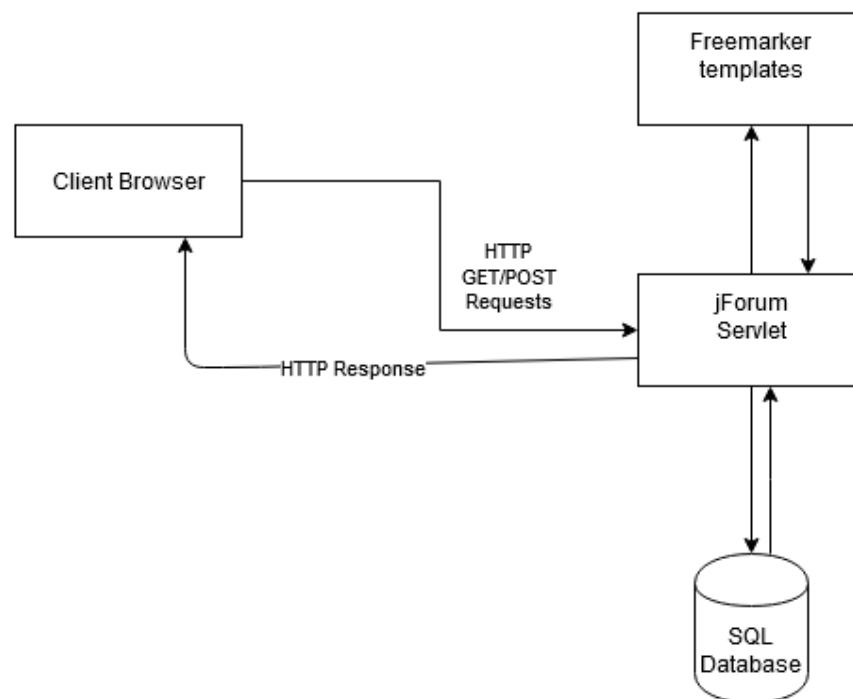
Task	Sub tasks	Task Delegation	Hour allocation / person
Zoom discussion meeting for team charter and finalising the team charter on 10-03-2020 at 11:00	-	All	2 hr (Completed)
Going over the Jforum documentation	-	All	2 hr (Completed)
Understanding Java Security Architecture.  Initially all of us went through the architecture, later on we divided the sections to get into details and we came together to share our understanding and come up with the architecture.	SQL Databases	Ranjana	2 hr 30 min (Completed)
	Data flow	Oskar	
	Client Browser	Efstathios	
	Freemarkers Template	Justas	
	jForum Servlet	Aimé	
Identifying important security aspects of Jforum	-	All	2 hr (Completed)
Overview and discussion for Audit plan	-	All	2 hr (Completed)
Team meeting with Alan presenting all the findings on 10-05-2020 at 14:00	-	All	2 hr (Completed)
Documentation of the overall work on 10-06-2020 at 12:00	-	All	2 hr (Completed)
Finalising the documentation on 10-06-2020 at 2:30		All	30 min (Completed)
Working on detection of Preliminary Vulnerability based on the preliminary report and Documentation of the findings	SQL Injection OS injection Sensitive Data Protection Credential management	Ranjana	10 hr (Completed)
	Inspection of input validation. Checking for vulnerable dependencies and scanning code	Oskar	
	Insufficient Access Control Mechanism, Error Handling	Efstathios	

	and Logging		
	XSS Cross Site Scripting Cookies & Session Hijacking SSO	Justas	
	Malicious and incorrect HTML Tags Handling special characters	Aimé	
Setting up, running Static analysis tool and Documentation of the analysis' results.	Coverity	Ranjana, Justas, Efstathios	8 hr
	Fortify	Oskar, Justas, Efstathios, Ranjana, Aimé	
	Dependency Checker	Oskar	
Final documentation	-	All	4 hr

# Jforum Architecture

## Data flow between the modules

User calls jForum with a GET or POST request to the `HttpServletRequest`. The jForum process that requests according to the logic in the java program. During this process data is read and written to the SQL database. The jForum returns results by providing values to variables in Freemarker templates, generating Web pages on the fly and returning them to the client browser. The data flow is presented in the diagram below:



*Figure 1 jForum Architecture*

Let us look into each module in detail

### 1. Client Browser

Client Browser is the medium/start point which is used to connect and perform users' actions to Jforum Servlet and receive a response through Freemarker's template interface from a user's perspective.

When a user requests a web page the http request is sent to the server. Based on the url, the request will start different servlets. This mapping between URL and servlets are done in the web.xml file on the server. I.e. the url pattern “/install/install.page” will start the install servlet and other URL: s will instead start the jForum servlet.



The included parameters in the request will then be passed on to the servlet, where the parameters 'module' and 'action' are used to instruct the servlet which actions to take.

## 2. Freemarker templates

The java servlet fetches one of the freemarker templates and assigns values to the parameters in the template, creating a complete html page. Then the servlet sends this as an http response to the client.

Freemarker templates are used in HTML Web pages to give users a more personal experience ie.: {Hello \${var1}!} -> would appear as {Hello Justas!}" after login into jForum server.

## 3. jForum Servlet

Java Servlet is a Java class that creates dynamical data within a HTTP server. This data is often presented in HTML or XML format, but it can also be presented in any other format for web browsers.

Multiple Servlets can run dynamically on the web server at the same time and allows the extension of its functions (access to databases, e-commerce transactions, etc.). jForum Servlet can be any compatible servlet container running on Java 8, such as the Apache Tomcat servlet. (*JForum2 Summary / JForum*, 2020)

Methods are defined by the servlet interface to initialize a servlet, to receive and respond to client requests, and to destroy a servlet and its resources. These are life-cycle methods.

Basic application properties are obtained using *javax.servlet.ServletConfig*, it returns a data servlet configuration object.

Jforum servlet session is initialized with a servlet configuration object using *init(ServletConfig config)* method, during the servlet life-cycle, the application can access startup information and basic information about itself.

Servlet access request and response are achieved using *javax.servlet.http.HttpServletRequest* and *javax.servlet.http.HttpServletResponse*.

The servlet's life-cycle is ended using the *destroy()* method, it destroys the servlet and cleans all the resources used. (*Interface javax.servlet.Servlet / Oracle Documentation*, 2020)

## 4. SQL Databases

Underlying database is the location where all the data/information of the application is stored. The database used for this assignment is mysql 5.6.10. The database is accessed via jForum Servlet using mysql client. The database has 36 tables with all the information from jForum

stored here including username, password, forum posts. Since all the sensitive to non-sensitive information of jForum is stored in the database, this module acts as a vulnerable point from attacks.

Database is connected using the database driver via `DataAccessDriver.java`, based on the presence of the database to be used, this class calls different access driver with `MySQL32DataAccessDriver.java` for mysql, `OracleDataAccessDriver.java` for Oracle and so on. By using the database driver, database connection is performed in `DBConnection.java` file. The connection has a try and catch section in case the connection is not successful. Now the connection is created, the SQL statements are all created in one file `generic_queries.sql` with variables assigned to each query. The assigned variable is used to access the SQL query. For example, listing member list is defined in `generic_queries.sql` into variable `UserModel.selectAll`, which is called in `GenericUserDAO.java` class. For each database execution, `DatabaseException.java` class is called to keep track of error messages. Beside for three places, all the sql queries are parameterized which works better in defying SQL Injection.

## 3. Vulnerabilities

### Improper Input Validation

The jForum application seems to utilize input validation on user input data. However, at some places the input validation is not sufficient. The main problem in many places is that the input validation is based on black-listing and not whitelisting. An approach not recommended as it might be easy to bypass at the same time as it can prevent some authorized inputs. (*Input Validation - OWASP Cheat Sheet Series*, 2020)

#### 1. User registration lacking proper server-side input validation of email addresses (CWE 20 & CWE79):

When inspecting the source code in net.jforum.view.forum.UserAction.java, we noticed that several checks are made on the username parameter inputted when a new user register. However, the input from the email address field is not validated. This makes it possible to input whatever value in the field and get it stored in the database, presented to other users visiting the forum. At the same time, this data will be used when the system tries to send emails to the user.



```
249= public void insertSave()
250 {
251     UserSession userSession = SessionFacade.getUserSession();
252     int userId = userSession.getUserId();
253
254     if ((SystemGlobals.getBoolValue(ConfigKeys.REGISTRATION_ENABLED)
255         && !SecurityRepository.get(userId).canAccess(SecurityConstants.PERMISSION_ADMINISTRATION))
256         || ConfigKeys.TYPE_SSO.equals(SystemGlobals.getValue(ConfigKeys.AUTHENTICATION_TYPE))) {
257         this.registrationDisabled();
258         return;
259     }
260
261     User u = new User();
262     UserDao dao = DataAccessDriver.getInstance().newUserDAO();
263
264     String username = this.request.getParameter("username");
265     String password = this.request.getParameter("password");
266     String email = this.request.getParameter("email");
267     String captchaResponse = this.request.getParameter("captchaResponse");
268
269     boolean error = false;
270     if (username == null || username.trim().equals("")
271         || password == null || password.trim().equals("")) {
272         this.context.put("error", I18n.getMessage("UsernamePasswordCannotBeNull"));
273         error = true;
274     }
275
276     if (username != null) {
277         username = username.trim();
278
279         if (error && username.length() > SystemGlobals.getIntValue(ConfigKeys.USERNAME_MAX_LENGTH)) {
280             this.context.put("error", I18n.getMessage("User-usernameTooBig"));
281             error = true;
282         }
283
284         if (error && username.indexOf('<') > -1 || username.indexOf('>') > -1) {
285             this.context.put("error", I18n.getMessage("User-usernameInvalidChars"));
286             error = true;
287         }
288
289         if (error && dao.isUsernameRegistered(username)) {
290             this.context.put("error", I18n.getMessage("UsernameExists"));
291             error = true;
292         }
293
294         if (error && dao.findByEmail(email) != null) {
295             this.context.put("error", I18n.getMessage("User-emailExists", new String[] { email }));
296             error = true;
297         }
298     }
```

Figure 2 Code section which lacks input validation

The only check made on the email is if the email already exists in the database, but no other validation is done.

To test this, we tried to register a new user, providing an email address value including tags and metacharacters. This produced an error message stating that the email address was not being valid.

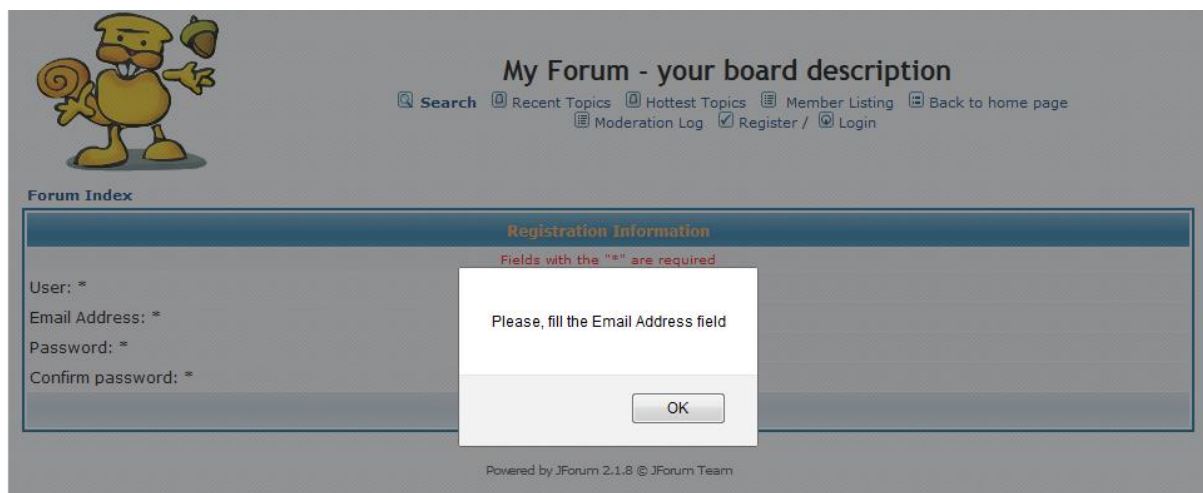


Figure 3 Error message from invalid tags in email address

As we had seen no input validation in the inspected source code, we suspected that the validation was done only on the client side. By inspecting the source code of the presented web page, we confirmed that the email was validated using a javascript on the client side, meaning it could easily be avoided by modifying the post request sent to the server.

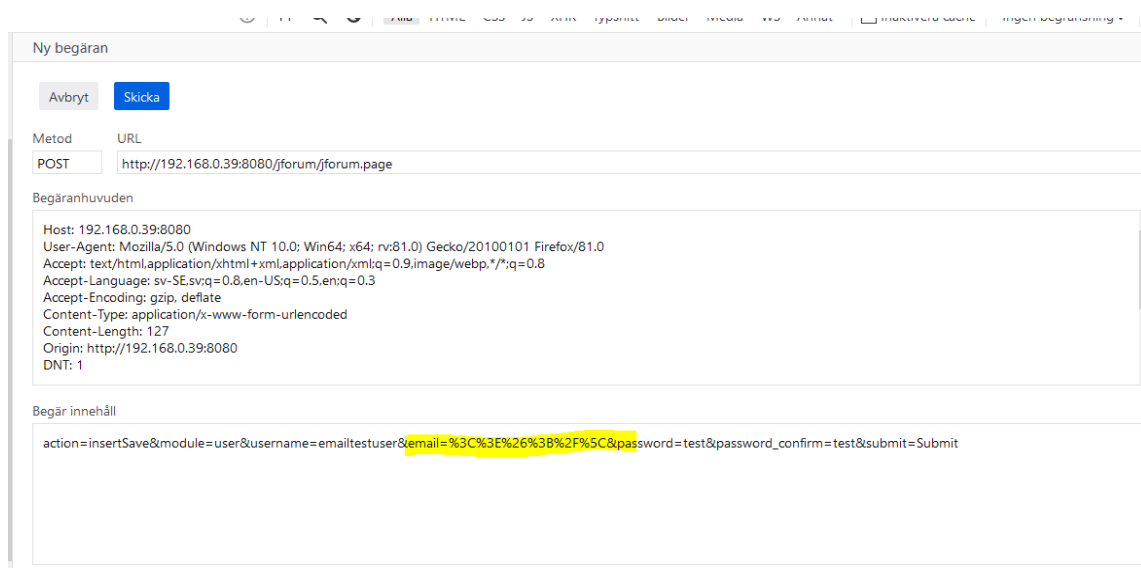


Figure 4 Source code inspection in web page

With this modified POST request anything can be added as an email address which will be stored in the database, presented to other users as well as been used when the jForum will try to send an email to the user.

## Consequences

This vulnerability could be used to add links to malicious web pages that the user will be redirected to when clicking the users email button (CWE-601) (*CWE - CWE-601: URL Redirection to Untrusted Site ('Open Redirect')* (4.2), 2020)

As this email address is both stored in the databases and used as input values for other functions such as the function sending emails, it could be used to exploit potential vulnerabilities in those functions.

## Resolution:

This vulnerability would be solved by implementing proper input control whitelisting characters used in email addresses as well as the format of an email address such as [aaa@bbb.com](mailto:aaa@bbb.com) (*Input Validation - OWASP Cheat Sheet Series*, 2020)

### 1. Forum file attachment validation does not follow recommended best practices:

By inspecting the code in `net.jforum.view.common.AttachmentCommon.java`, responsible for checking file attachments in forum posts we noticed that it seems to only check against file extensions configured as explicitly forbidden, so called black-listing. This is not a recommended approach as there will be a need to carefully configure this list. For example, if forbidding php files, all other file extensions that would be run in the same way.

## Consequences:

The lack of white-listing capabilities for the file attachment makes it hard to ensure that no harmful files can be uploaded. This increases the risk of the CWE-184 vulnerability - Incomplete List of disallowed Inputs (*CWE - CWE-184: Incomplete List of Disallowed Inputs* (4.2), 2020). This also creates a risk of file injections (*Unrestricted File Upload | OWASP*, 2020).

## Solution:

If the checks on the files were instead checked against a white-list of explicitly allowed file types the problem of Incomplete List of disallowed Inputs (CWE-184) would be less likely (CWE - CWE-184: *Incomplete List of Disallowed Inputs* (4.2), 2020).

### 2. User information update form does not validate all input and as a result, a stored XSS could be implemented (CWE 20 & CWE79):

When inspecting the source code in `net.jforum.view.forum.UserCommonJava`, we noticed that most of the data inputted as user personal information where validated through the `safeHtml` function to remove harmful tags, such as i.e. javascript. However, this was missed for the user biography information, meaning any input could be valid including harmful javascripts.



```
02
03     SafeHtml safeHtml = new SafeHtml();
04
05     u.setId(userId);
06     u.setEmail(safeHtml.makeSafe(request.getParameter("email")));
07     u.setIcq(safeHtml.makeSafe(request.getParameter("icq")));
08     u.setAim(safeHtml.makeSafe(request.getParameter("aim")));
09     u.setMsn(safeHtml.makeSafe(request.getParameter("msn")));
10     u.setYim(safeHtml.makeSafe(request.getParameter("yim")));
11     u.setFrom(safeHtml.makeSafe(request.getParameter("location")));
12     u.setOccupation(safeHtml.makeSafe(request.getParameter("occupation")));
13     u.setInterests(safeHtml.makeSafe(request.getParameter("interests")));
14     u.setBiography(request.getParameter("biography"));
15     u.setSignature(safeHtml.makeSafe(request.getParameter("signature")));
16     u.setViewEmailEnabled(request.getParameter("viewemail").equals("1"));
17     u.setViewOnlineEnabled(request.getParameter("hideonline").equals("0"));
18     u.setNotifyPrivateMessagesEnabled(request.getParameter("notifypm").equals("1"));
19     u.setNotifyOnMessagesEnabled(request.getParameter("notifyreply").equals("1"));
20     u.setAttachSignatureEnabled(request.getParameter("attachsig").equals("1"));
21     u.setHtmlEnabled(request.getParameter("allowhtml").equals("1"));
22     u.setLang(request.getParameter("language"));
23     u.setBbCodeEnabled("1".equals(request.getParameter("allowbbcode")));
24     u.setSmiliesEnabled("1".equals(request.getParameter("allowsmilies")));
25     u.setNotifyAlways("1".equals(request.getParameter("notify_always")));
26     u.setNotifyText("1".equals(request.getParameter("notify_text")));
27
28     String website = safeHtml.makeSafe(request.getParameter("website"));
29     if (!StringUtil.isEmpty(website) && !website.toLowerCase().startsWith("http://"))
30         website = "http://" + website;
31 }
32
```

Figure 5 Code Section for Biography request definition

To test this, we logged in to the running jForum website and tried to edit the user biography, inputting javascript code.

Current Password: \*  
You only need to enter the current password if changing to a new password or changing your e-mail address.

New Password: \*  
You only need to enter a new password if changing it.

Confirm password: \*  
You only need to enter a new password if changing it.

**General Information about yourself**

This information will be publicly viewable

ICQ UIN: XrBKipatXrBKipa

AIM Info: CXoyrXIICXoyrXIICagsBNaz

MSN Messenger: ifOMGEnUifOMGEnUWjaTmuyJ

Yahoo Messenger: dVAvGmbidVAvGmbiLVGFeqdB

Web Site: http://BtLCpbPpBtLCpbPpqcFpubF

Location: XAeBxlHTXAeBxlHTEkewWfhp

Occupation: MBNshdYJMBNshdYJwlhMmQLX

Interests: ibUXCBZpibUXCBZpQaAfPJQT

Biography: `<button onclick="myFunction()">Click me</button>  
<p id="demo"></p>  
<script>  
function myFunction() {  
 document.getElementById("demo").innerHTML =  
 "Hello World";  
}  
</script>`

Signature:  
This is a text block that will be added to your message's end. Optional, limit of 255 chars

**Preferences**

Figure 6 Insertion of HTML tags in Biography

Information gets stored, without any changes. To explore the effects of this we navigated to the user information presented to other users. As suspected, we could see the javascript code in the biography section executed as intended.

**Profile for :: asdf**

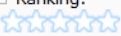
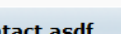

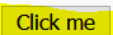
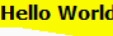
Avatar	All about asdf
 Ranking:  Karma: 	Registration date: <b>10/10/2020 14:09:00</b> Number of messages posted: <b>[36] Messages posted by asdf</b> Created topics: No topic created From: <b>XAeBxlHTXAeBxlHTEkewWfhp</b> Website: <b>http://BtLCpbPpBtLCpbPpqcFpubF</b> Occupation: <b>MBNshdYJMBNshdYJwlhMmQLX</b> Interests: <b>ibUXCBZpibUXCBZpQaAfPJQT</b>
<b>Contact asdf</b> Email Address:  email Private Message:  pm MSN Messenger: ifOMGEnUifOMGEnUWjaTmuyJ Yahoo Messenger: dVAvGmbidVAvGmbiLVGFeqdB ICQ UIN:  ICQ	Biography:   My Bookmarks: There are no bookmark entries for this user.

Figure 7 Biography result after adding html tags

## Consequences

This error, which is caused by lack of proper input validation, can be exploited by any registered user to conduct a stored cross-site scripting attack (CWE-79) (CWE - CWE-79: *Improper Neutralization of Input During Web Page Generation ('Cross-Site Scripting')* (4.2), 2020) According to OWASP this can cause a number of problems for the end-user being victim of the XSS attack, where the most severe involve getting a session hijacked. But it can also be used to disclose end user files. (*Cross Site Scripting (XSS) Software Attack | OWASP Foundation, 2020*)

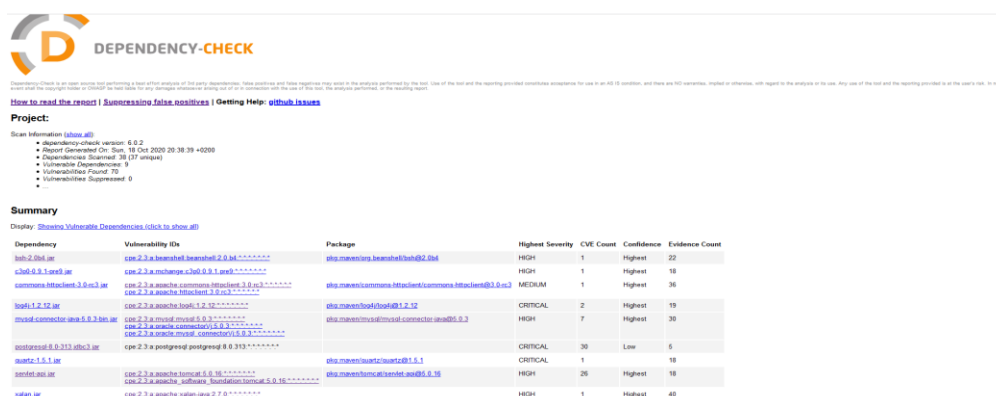
## Resolution

A solution to this would be to add proper input validation and sanitize html code that is inputted to the biography field before it is stored and presented to other users. OWASP recommends using e.g. `HtmlSanitizer` or `OWASP Java HTML Sanitizer`. (*Cross Site Scripting Prevention - OWASP Cheat Sheet Series, 2020*)

## Dependencies

The jForum application depends on several outdated libraries with known vulnerabilities. Among the libraries used, there are published vulnerabilities in Log4j, Mysql-connector-java, postgresql-8.0-313.

When analysing the code with OWASP Dependency check it showed a lot of vulnerabilities in several more of the dependencies of jForum, spanning from cross-site scripting to DoS-attacks and unauthorized access to the database. (*OWASP Dependency-Check Project | OWASP, 2020*)



The screenshot displays the OWASP Dependency-Check web interface. At the top, there's a logo and the text 'DEPENDENCY-CHECK'. Below this, a 'Project:' section shows 'jforum' with a list of scan information: 'dependency-check version: 6.6.2', 'Report Generated On: Sun, 18 Oct 2020 20:38:39 +0200', 'Dependency-Check Version: 3.0.1 (unpkg)', 'Vulnerable Dependencies: 9', 'Vulnerabilities Found: 70', and 'Vulnerabilities Suppressed: 0'. A 'Summary' section follows, with a table listing the vulnerable dependencies. The table has columns for 'Dependency', 'Vulnerability IDs', 'Package', 'Highest Severity', 'CVE Count', 'Confidence', and 'Evidence Count'.

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
log4j-2.10.0.jar	cve-2.3.x, log4j-2.10.0, log4j-2.10.0	org.apache.logging.log4j:log4j-2.10.0	HIGH	1	Highest	22
commons-httpclient-3.0.1.jar	cve-2.3.x, commons-httpclient-3.0.1, commons-httpclient-3.0.1	commons-httpclient:commons-httpclient-3.0.1	MEDIUM	1	Highest	36
mysql-connector-java-5.1.30.jar	cve-2.3.x, mysql-connector-java-5.1.30, mysql-connector-java-5.1.30	mysql:mysql-connector-java-5.1.30	CRITICAL	2	Highest	19
postgresql-8.0-313.jdbc3.jar	cve-2.3.x, postgresql-8.0-313, postgresql-8.0-313	org.postgresql:postgresql-8.0-313	HIGH	7	Highest	30
commons-httpclient-3.0.1.jar	cve-2.3.x, commons-httpclient-3.0.1, commons-httpclient-3.0.1	commons-httpclient:commons-httpclient-3.0.1	CRITICAL	30	Low	5
commons-httpclient-3.0.1.jar	cve-2.3.x, commons-httpclient-3.0.1, commons-httpclient-3.0.1	commons-httpclient:commons-httpclient-3.0.1	CRITICAL	1	Low	18
commons-httpclient-3.0.1.jar	cve-2.3.x, commons-httpclient-3.0.1, commons-httpclient-3.0.1	commons-httpclient:commons-httpclient-3.0.1	HIGH	26	Highest	18
commons-httpclient-3.0.1.jar	cve-2.3.x, commons-httpclient-3.0.1, commons-httpclient-3.0.1	commons-httpclient:commons-httpclient-3.0.1	HIGH	1	Highest	40

Figure 8 8 Dependency check via OWASP dependency check software



## Consequences

According to OWASP, vulnerabilities in dependency libraries can have severe impacts. The vulnerabilities can often be exploited, often using already-written exploits. It might be specifically problematic when the vulnerable component runs with the same privileges as the application depending on it. (A9, 2020).

There is no clear answer to which vulnerabilities that are created by vulnerable dependencies, as there could be whichever vulnerability in a dependency library. However, the vulnerability report we got from running the OWASP dependency checker showed vulnerabilities including arbitrary code execution and billion laughs attack. It corresponds to CWE-776 (CWE - CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')) (4.2), 2020)

## Resolution:

Many of the found vulnerabilities in these libraries are fixed in newer versions. Therefore, a solution would be to make sure to use the latest versions of the libraries. To tackle the problem of vulnerable dependencies, OWASP recommends employing good patch management. (A9, 2020)

## SQL Injection

Database is the backend of jForum. By bypassing the application, one can connect to the database and execute malicious SQL statements from select, insert to even truncate. SQL Injection can occur by simply not performing proper input validation to using concatenation on input selection. The SQL Injections' main threat is access to unauthorized sensitive data. (*What Is SQL Injection (SQLi) and How to Prevent Attacks*, 2020)

With most queries being parameterized, following three sections were identified as possible sections which can be exploited for SQL Injections

### 1. <http://localhost:8080/jforum/user/lostPassword.page>

The section to retrieve email by username uses straightforward SQL making it susceptible for SQL Injections.



Upon entering “ into updating new text, following error message is thrown, meaning neither input validation is done nor predefined statement is used:

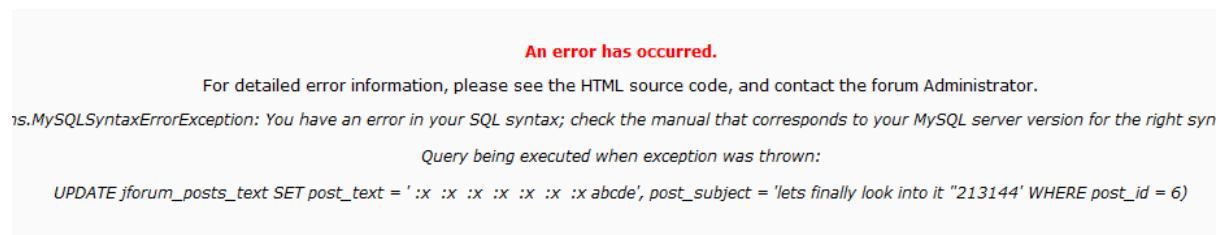


Figure 12 Result after update tries in forum text

## Consequence

The consequences of SQL Injection can vary from authentication bypass to gaining administrative privileges. Because of the input validation flaw, Sql injection can be performed to gain access to unauthorized data as well as perform unauthorized manipulation of data. Using the SQL injection, an attacker can impersonate the user by gaining the access of credentials as well. The data manipulation can be hazardous as attackers can remove the whole data from the database disrupting the website. (*Consequences of SQL Injection*, 2020) (*Biggest Threat to Application Security: SQL Injection Attacks*, 2020)

## Resolution

In both sections, we can see that the statement is created in the execution section and input is directly added in the SQL injection. One of the ways to rewrite such queries is to add a section of input validation, i.e., before passing the input to the query, the inputs are validated along with clearing of special characters. This way Sql injection via bypass of special characters is not possible. This aligns with **CWE-89** which deals with SQL injection via improper neutralization of special elements (*CWE - CWE-89: Improper Neutralization of Special Elements Used in an SQL Command ('SQL Injection')* (4.2), 2020).

Thus, it can be resolved by creating a separate execution domain instead of input validation and execution pattern occurring at the same section which maps to Execution Domain Pattern of JAVA Security Architecture. (Rosado et al., 2006)

## OS Injection

On the preliminary vulnerability assessment, we were not able to identify the OS injection. The code inspection showed one OS command being executed is mkdir. This command follows

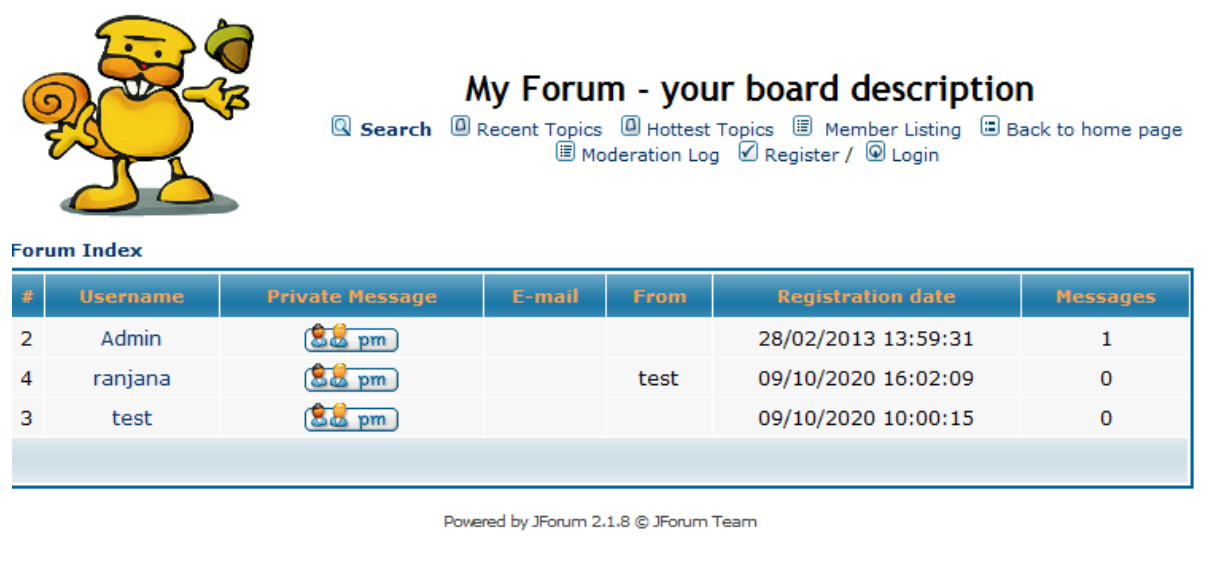
OWASP suggestion by usage of built-in library for file handling. This is supported by static analysis via both Fortify and Coverity, i.e., both the tools did not detect any OS injection in jForum. (*OS Command Injection Defense - OWASP Cheat Sheet Series*, 2020)

## Sensitive Data Protection

Inadequate protection of sensitive information can cause sensitive data exposure. With privacy policies and GDPR, exposure of sensitive data is an extremely severe vulnerability. The exposure can occur simply with lack of proper encryption to exposure on data which could be used for attack, example exposure of username making the login credentials susceptible to brute force attack. (A3, 2020)

The exposure of username is risking jForum to sensitive data exposure. Member listing is possible even without login which means one can get username even without account.

<http://localhost:8080/jforum/user/list.page;jsessionid=9EB3100FEC612EF5061BC2B39BD6ACE9>




#	Username	Private Message	E-mail	From	Registration date	Messages
2	Admin				28/02/2013 13:59:31	1
4	ranjana			test	09/10/2020 16:02:09	0
3	test				09/10/2020 10:00:15	0

Figure 13 Output shows username without login

Failed login was tried 100 times and actual password was entered in the end, which worked as a normal login rather than locking the account. The credentials are stored as single md5 hash, making them vulnerable. Lack of restriction in failed login has created a risk of sensitive data exposure.

## Consequence

Since the login section does not have restrictions on how many times failed login can be done, the login details are at risk of brute force attack. Possibility of brute force attack means that one's account can be hacked within a possible amount of time. Lack of salting aids in Brute force attack as well.

## Resolution

Unless necessary, sensitive data should not be shared with unauthorized users. Proper encryption algorithms should be enforced in password to ensure brute force attack timing is increased. Since MD5 has already been broken, a stronger hashing algorithm should be used. Sensitive Data Protection falls under CWE-326 (*CWE - CWE-326: Inadequate Encryption Strength (4.2)*, 2020) and CWE-359 (*CWE - CWE-359: Exposure of Private Personal Information to an Unauthorized Actor (4.2)*, 2020).

## Credential Management in Broken Authentication

Credentials is sensitive data that requires extra protection as one can perform man in the middle attack if provided with credentials. Upon jForum code inspections, following scenarios were identified which proved jForum lacks a proper credential management.

### 1. Predictable Login Credentials

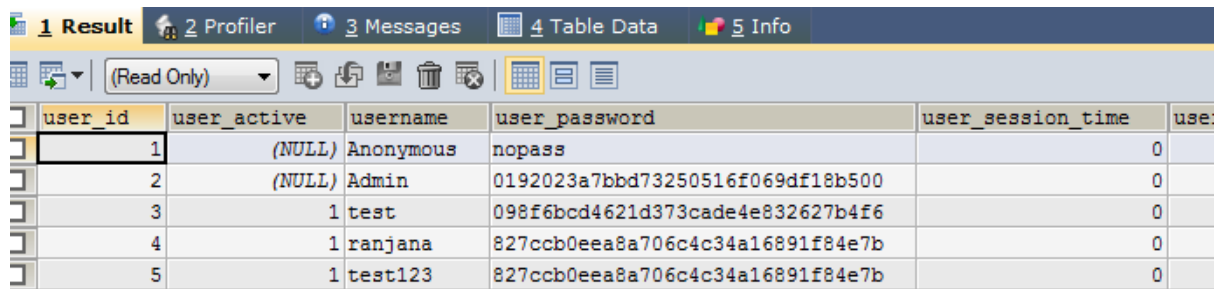
The credentials creation in jForum does not have password policy, meaning entering 12345 as password is acceptable without any warnings. This means that websites can have usernames and passwords which are easy to guess by attackers creating a vulnerability and lack of proper policy (*Broken Authentication and Session Management / Hdiv Documentation*, 2020).

### 2. Brute Force attack

Lack of retry restriction in forget password made jForum susceptible to Brute Force attack (*Broken Authentication and Session Management / Hdiv Documentation*, 2020).

### 3. Lack of Salt in hashing Passwords

Absence of salt means if two users have the same password then upon checking the database, one can easily identify such password (*Broken Authentication and Session Management / Hdiv Documentation*, 2020). Thus, only md5 hash of password has risked Jforum of insider attack. Figure 14 below shows last two credentials to have same md5 as they had same password:



The screenshot shows a database tool interface with a table containing user information. The table has columns for user\_id, user\_active, username, user\_password, and user\_session\_time. The data rows show users with various usernames and MD5 hashed passwords.

user_id	user_active	username	user_password	user_session_time
1	(NULL)	Anonymous	nopass	0
2	(NULL)	Admin	0192023a7bbd73250516f069df18b500	0
3	1	test	098f6bcd4621d373cade4e832627b4f6	0
4	1	ranjana	827ccb0eea8a706c4c34a16891f84e7b	0
5	1	test123	827ccb0eea8a706c4c34a16891f84e7b	0

Figure 14 md5 hashes of passwords in database

## Resolution

Proper password policy must be defined for credential management to reduce the risk from Brute force attack. Salting must be enforced while storing passwords so that one cannot identify if two passwords are the same. Account lock must be implemented as well to reduce risk of Brute force attack as simple as blocking the user in three failed logins. Credential Management falls under CWE-326 (CWE - CWE-326: *Inadequate Encryption Strength* (4.2), 2020) and follows the attack pattern Keytool (Dougherty, 2009).

## Insufficient Access Control Mechanism

RBAC is not properly implemented as when a user logs-in, a role is assigned by default and there is no further authentication process. The role is defined in both code as well as database, but when trying to assign a role from the front-end, it is not possible both via admin account and user account. The arguments could be verified from the screenshot below that was taken from the Admin Control Panel as the assignment of roles was not possible. Since by default, a role is assigned, and no proper role designation is possible RBAC is not fully implemented here.



Figure 15 Admin Panel with role listing

## Consequence

An insufficient implementation of the Access Control Mechanism can lead to severe consequences such as elevation of privilege (e.g. acting as an admin when logged in as a normal user). Furthermore, unauthorized elevation of privileges can lead to sensitive data disclosure, modification or destruction of data and performing different business functions outside of user's permissions (*OWASP Top Ten Web Application Security Risks / OWASP*, 2020)

## Resolution

The vulnerability could be fixed by implementing the security design principle of “*least privilege*” in the ACM, by using the appropriate authentication process of users and by assigning the designated role to each of them. Furthermore, the RBAC pattern could be used as a complementary source of resolving the issue as it constitutes a detailed guide for an effective RBAC (Rosado et al., 2006). The vulnerability corresponds to CWE 284/285 weakness (CWE - CWE-284: *Improper Access Control* (4.2), 2020) (CWE - CWE-285: *Improper Authorization* (4.2), 2020)

## Malicious and incorrect HTML tags

It seems that input content validators are missing or can be bypassed, as input is not properly checked. Malicious HTML tags can be injected in the web page. As seen in the XSS and SQL injection parts, scripts can be injected and executed. We suspect the presence of incorrect HTML tags that will create vulnerabilities and give access to malicious HTML injections.

## Consequences

As mentioned in the SQL injection part, SQL Injections allow users to gain elevated privileges and grant access to unauthorized data as well as perform unauthorized manipulation of data (*Consequences of SQL Injection*, 2020)(*Biggest Threat to Application Security: SQL Injection Attacks*, 2020).

In addition, according to the XSS injections part, XSS Injections can create end user issues, from Annoyance to Sensitive Data disclosure (*Team*, 2019). The vulnerability corresponds to CWE-80 (CWE - CWE-80: *Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)* (4.2), 2020).

## Resolution

To avoid attacks such as SQL and XSS injections, adding input validation to handle malicious content is one solution, i.e, check if the input is valid first, handling special characters, it makes SQL and XSS injection via bypass of special characters or malicious content not possible.

## Handling special characters

The characters “<>” are not properly handled: It allows XSS injections. They are checked for the user registration part, but there is no handling for user information input in the user biography input form. The characters “ ‘ ” are not properly handled: It allows SQL injections. It seems vulnerabilities are detected because ‘ and < characters are not properly handled, jForum is by this way vulnerable and unsafe.

We suspect vulnerabilities in the absence of validators and white list of allowed characters to handle special characters input, as well as the handling of malicious content in general.

## Consequences

Since special characters are not properly handled, it creates possibilities for special characters combination and SQL or XSS injections.

As example for special characters combination:

An application which executes almost everything which is passed to it from the current terminal by the user without sanitizing and blocking user input. If the application doesn't implement



appropriate signals handling, we may interrupt or suspend program execution by sending respectively Ctrl+C (^C) or Ctrl+Z (^Z) combinations. (*Custom Special Character Injection / OWASP, 2020*). The vulnerability corresponds to CWE-138 (*CWE - CWE-138: Improper Neutralization of Special Elements (4.2)*, 2020).

## Resolution

The malicious content sending can be prevented by checking input to handle special characters. For this purpose, it is necessary to consider the implementation of validators and white lists.

## XSS Cross-Site Scripting

After creating a user account on jForum and adding a simple script:

```
*/ <script>alert('Hello I am hacked') </script> /* ,
```

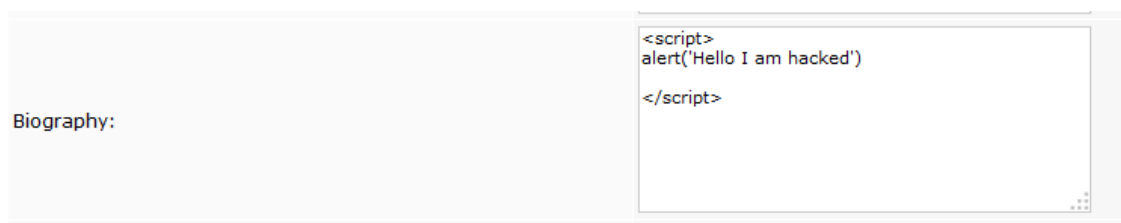


Figure 16 Adding javascript in biography

In the “Biography” field of users’ profile, we were able to expose a XSS vulnerability and technically compromise users' experience with the application. As a result, a more advanced script would cause more serious damage. The location of the issue was found in UserCommon.java where invalidated input is located.

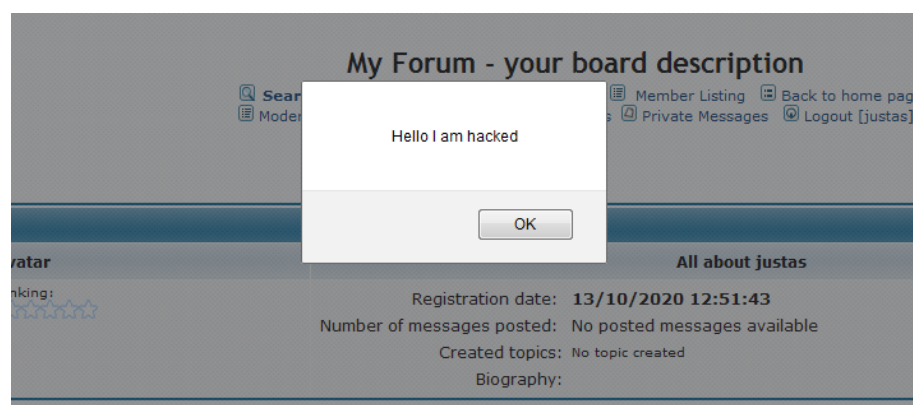


Figure 17 Result after adding Javascript in Biography

## Consequence

XSS can cause a variety of problems for the end user which can lead to:

- Annoyance
- Session Hijacking
- Disclosure of Sensitive Data
- Cross Site Request Forgery Attacks

(*Cross-Site Scripting / What Is XSS Attack? / Netsparker, 2019*)

## Resolution

OWASP Team suggests a couple ways to prevent XSS:

- Turn off HTTP TRACE support on all web servers.
- Never insert untrusted data except in allowed locations.
- HTML encode before inserting untrusted data into html element content
- Attributes encode before inserting untrusted data into HTML common attributes.
- JavaScript encode before inserting untrusted data into Javascript data values

(*Cross Site Scripting Prevention - OWASP Cheat Sheet Series, 2020*)

The vulnerability corresponds to CWE-79 (*CWE - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-Site Scripting') (4.2), 2020*)

## Cookies and Session Hijacking

Using a debugger Firebug 1.11.2 we checked the cookie management and found JSESSIONID cookie gives the exact value of the ID, which can be used for Session Hijacking and jForumUserId array list is exposed due to improper and poor coding.

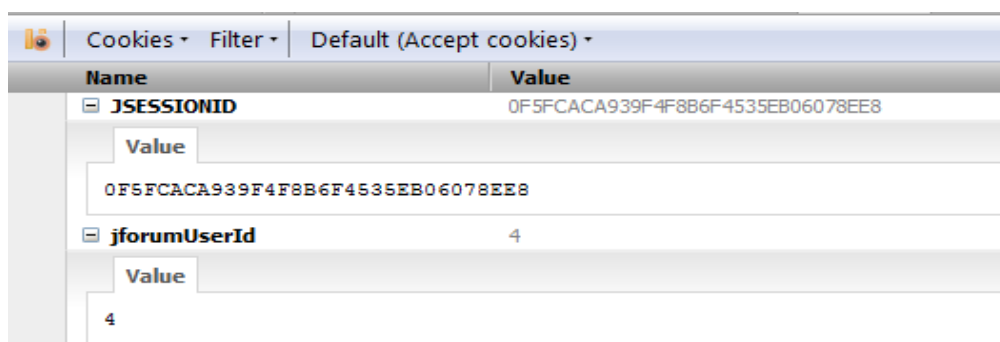


Figure 18 Cookies showing userid

```

- public User(int userId)
  {
    this.id = userId;
  }

- /**
  * Default Constructor
  */
- public User()
  {
    this.groupsList = new ArrayList();
  }

```

Figure 19 Cookies definition code with userid definition

Code can be found in net.jforum.entities.User.java

After running static analysis tools this issue with cookie mishandling appeared in different lines of code - ControllerUtils.java(364) and was explained by Header Manipulation Vulnerability.

## Consequence

Poor cookie handling can cause attacks such as:

- Man-in-the-middle attack.
- Session Sniffing.
- Client-side attacks (XSS, Trojans, malicious JavaScript Code executions, etc.).

(*Session Hijacking Attack Software Attack / OWASP Foundation, 2020*)

## Resolution

OWASP provides a couple ways of handling such issue by following these steps:

- Bind sessions to IP addresses.
- Invalidate (unset cookie, unset session storage, remove traces) of a session whenever a violation occurs.
- roll session ID whenever elevation occurs (e.g when a user logs in, the session ID of the session should be changed, since its importance is changed.).
- Encrypt data traffic between the parties using SSL/TLS.
- Invalidate the Session id after user login.
- Set a session expiration time.
- Set an inactivity timeout.

(Woschek, 2015) (*Input Validation - OWASP Cheat Sheet Series, 2020*)

It corresponds to CWE-784 (*CWE - CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision (4.2)*, 2020)

## 4. Static Tool analysis

We performed static code analysis using both Fortify and Coverity. The analysis was performed by following the steps provided. After analysis, our preliminary vulnerability report was aligned with static tool analysis' results but also a significant number of unidentified vulnerabilities were found.

### Coverity

After checking the result from Coverity, we could detect quite more vulnerabilities and it provided us with a guide as per where to look for vulnerability as well. Few of the interesting ones detected during static code analysis were Divide from zero, this was detected 16 times by Coverity while we were not able to detect nor suspect at all during preliminary analysis. In a similar manner, resource leak was high in number meaning attackers could trigger resource leak and cause Denial of Service attack. (*Unreleased Resource* / OWASP, 2020)

```
Analysis summary report:
-----
Files analyzed           : 460 Total
  JSP                    : 3
  Java                   : 335
  Java (without build)   : 26
  JavaScript             : 60
  Text                   : 36
Total LoC input to cov-analyze : 44590
Functions analyzed       : 11257
Classes/structs analyzed : 505
Paths analyzed          : 561511
Time taken by analysis   : 00:02:52
Defect occurrences found : 114 Total
  14 BAD_LOCK_OBJECT
  3 CALL_SUPER
  2 CHECKED_RETURN
  1 CONFIG.MISSING_GLOBAL_EXCEPTION_HANDLER
  1 CSRF
  16 DIVIDE_BY_ZERO
  1 DOM_XSS
  6 FORWARD_NULL
  1 GUARDED_BY_VIOLATION
  2 INSECURE_COMMUNICATION
  2 LOCK_EVASION
  1 MISSING_AUTHZ
  2 NON_STATIC_GUARDING_STATIC
  26 NULL_RETURNS
  2 OVERFLOW_BEFORE_WIDEN
  20 RESOURCE_LEAK
  1 RISKY_CRYPT0
  3 SQLI
  5 SWAPPED_ARGUMENTS
  1 TRUST_BOUNDARY_VIOLATION
  1 UNENCRYPTED_SENSITIVE_DATA
  1 UNLOGGED_SECURITY_EXCEPTION
  1 UNSAFE_REFLECTION
  1 WEAK_PASSWORD_HASH
Additional defects, SpotBugs : 38
Additional defects, JSHint   : 36
```

Figure 20 Result from Coverity

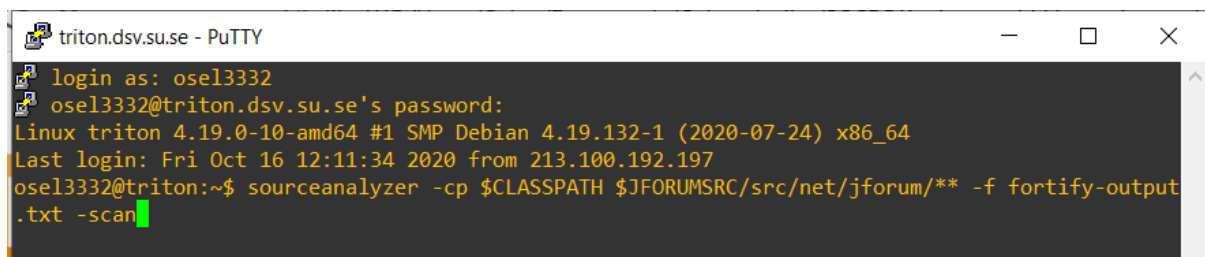
Usage of Coverity was straightforward and simple, as it was installed in Windows so there was no need to run any commands rather following simple instructions in the Application itself. The result was easy to understand as well since it provided where all the vulnerabilities are found and provided a comprehensive list of overall vulnerabilities. After locating the location of vulnerabilities in Coverity, code inspection became a guided code inspection. Following result from Coverity provides the overall vulnerabilities in the jForum code.

## Fortify

We performed a static analysis overall jforum folder with the aid of the Fortify tool. The static analysis results identified several vulnerabilities which were classified as low, high and critical risk. Furthermore, the vulnerabilities found through the manual code inspection were also identified by the tool alongside a significant number of new ones.

The usage of the Fortify tool was quite straightforward as we followed the instructions from the course staff to install and run the tests. We used the following command to execute the test on the whole source code location.

***\$JFORUMSRC/src/net/jforum/\*\****



```

triton.dsv.su.se - PuTTY
login as: osel3332
osel3332@triton.dsv.su.se's password:
Linux triton 4.19.0-10-amd64 #1 SMP Debian 4.19.132-1 (2020-07-24) x86_64
Last login: Fri Oct 16 12:11:34 2020 from 213.100.192.197
osel3332@triton:~$ sourceanalyzer -cp $CLASSPATH $JFORUMSRC/src/net/jforum/** -f fortify-output
.txt -scan

```

*Figure 21 Script executed to run code analysis in all folders*

The results were proven useful in our overall work to verify our manual inspection results, compared to Coverity's, and acquire a deeper insight in the Jforum's vulnerabilities. To conclude, vulnerabilities with high and critical risk identified on the results will be presented on the table below:

*Table 1 High to Critical vulnerabilities detected by Fortify*

Vulnerability	No. of hits	Type	Probability
Insecure Randomness	2	Semantic	High
Cookie Security	1	Overly Broad Path	High

Access control issues	1170	Database: Dataflow	High
Privacy violation	11	Dataflow	High/Critical
Header manipulation	96	Cookies: Dataflow/ SMTP: Dataflow	High
Server-Side Request Forgery	19	Dataflow	High
XML External Entity Injection	142	Dataflow	High
SQL Injection	316	Dataflow	High/Critical
Log Forging	240	Dataflow	High
Open Redirect	22	Dataflow	High
Path Manipulation	1203	Dataflow	High/Critical
Dynamic Code Evaluation	46	JNDI Reference Injection : Dataflow	High
Null Dereference	3	Controlflow	High
Unreleased Resource	11	Streams: Controlflow	High
Probability Flow	14	Locale Dependent Comparison : Controlflow	High
Code Correctness	4	Double-Checked Locking : Structural	High
Password Management	10	Password in comment Hardcoded Password	High

```
stathispsyllakis — ssh efp5309@triton.dsv.su.se — 171x61
Last login: Thu Oct 15 11:48:57 2020 from triton.dsv.su.se
efp5309@triton:~$ JFORUMSRC=/opt/sosec-source/jforum-src
efp5309@triton:~$ CLASSPATH=$JFORUMSRC/WEB-INF/classes/:$JFORUMSRC/lib/*.jar:$JFORUMSRC/lib/*.jar
efp5309@triton:~$ sourceanalyzer -cp $CLASSPATH $JFORUMSRC/src/net/jforum/JForum.java -f fortify-output.txt -scan
efp5309@triton:~$ less fortify-output.txt

[/opt/sosec-source/jforum-src/src/net/jforum]
(95F2D592B0B3D0DC7248CC98739B8D83 : low : System Information Leak : semantic ]
JForum.java(117) : Throwable.printStackTrace()
(0E41D96CB286D3C5E87F4E1F176A32B : low : System Information Leak : Incomplete Servlet Error Handling : structural ]
JForum.java(128)
(5F23FAABFA26E7E4DF813FED23098EA6 : low : Poor Error Handling : Throw Inside Finally : structural ]
JForum.java(207)
(F8C8BE1D007B62DF57BEA5E5B78FB8F3 : low : Poor Error Handling : Overly Broad Throws : structural ]
JForum.java(212)
(214EBD9879DA3F9781CA15FA52D1B1E9 : high : Code Correctness : Double-Checked Locking : structural ]
JForum.java(245)
SynchronizedBlock [JForum.java(244)]
IfStatement [JForum.java(243)]
(67CFAEA19AE0D1169B5A8A9962411F9 : low : Poor Error Handling : Overly Broad Catch : structural ]
JForum.java(257)
(BEEAE214DB14136C8D25B8091B24364 : low : Poor Error Handling : Empty Catch Block : structural ]
JForum.java(257)
(72FE5E863B4F977C43BA7F2C7EF21F25 : low : Dead Code : Expression is Always false : structural ]
JForum.java(281)
(323207ECD4A0C5A4DB51B8CB7C2AF09A : low : Poor Error Handling : Overly Broad Throws : structural ]
JForum.java(300)
(A866772E1DC7C8FC55E4DB32609D88A : low : Poor Logging Practice : Use of a System Output Stream : structural ]
JForum.java(310)
(E40291A7F21E8D7279FAB4435F65A87E : low : Poor Error Handling : Overly Broad Catch : structural ]
JForum.java(316)
(A61FDD8D109EEA7B0D5D03C6D8484445 : low : Poor Error Handling : Empty Catch Block : structural ]
JForum.java(316)
~
~
~
```

Figure 22 Result from Coverity

```
→ Bureau head -2 fortify-output.txt && cat fortify-output.txt | grep high -A1
[/opt/sosec-source/jforum-src/templates]
[3FB3C3948C3461BE06FDED989D76C3EE : high : Privacy Violation : Autocomplete : content ]
default/recover_password.htm(63)
--
[3FB3C3948C3461BE06FDED989D76C3EF : high : Privacy Violation : Autocomplete : content ]
default/recover_password.htm(69)
→ Bureau █
```

Figure 23 Highly critical result from Fortify

## OWASP Dependency checker

To check for vulnerable dependencies we used a tool called OWASP dependency checker, provided by OWASP. The tool scans for used dependency libraries and searches a number of vulnerability databases. The found vulnerabilities are then presented as an html page presenting which vulnerabilities found in which packages. (*OWASP Dependency-Check Project / OWASP*, 2020)



```

PS C:\Users\Oskar\Documents\Oskar studier\S0SEC\Assignment4> .\dependency-check-6.0.2-release\dependency-check\bin\de
pendency-check.bat --scan ".\jforum-src\jforum-src\**\*.jar"
[INFO] Checking for updates
[INFO] Skipping NVD check since last check was within 4 hours.
[INFO] Skipping RetireJS update since last update was within 24 hours.
[INFO] Check for updates complete (362 ms)
[INFO]


Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives
and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided const
itutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to th
e analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the co
pyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of thi
s tool, the analysis performed, or the resulting report.

[INFO] Analysis Started
[INFO] Finished Archive Analyzer (2 seconds)
[INFO] Finished File Name Analyzer (0 seconds)
[INFO] Finished Jar Analyzer (0 seconds)
[INFO] Finished Central Analyzer (0 seconds)
[INFO] Finished Dependency Merging Analyzer (0 seconds)
[INFO] Finished Version Filter Analyzer (0 seconds)
[INFO] Finished Hint Analyzer (0 seconds)
[INFO] Created CPE Index (3 seconds)
[INFO] Finished CPE Analyzer (7 seconds)
[INFO] Finished False Positive Analyzer (0 seconds)
[INFO] Finished NVD CVE Analyzer (2 seconds)
[INFO] Finished Sonatype OSS Index Analyzer (4 seconds)
[INFO] Finished Vulnerability Suppression Analyzer (0 seconds)
[INFO] Finished Dependency Bundling Analyzer (0 seconds)
[INFO] Analysis Complete (17 seconds)

```

Figure 24 Execution of OWASP dependency checker

Running this command, we generated the following output page presenting vulnerabilities in the dependencies of jForum.



Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies. False positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

[How to read the report](#) | [Suppressing false positives](#) | [Getting Help](#) | [Github Issues](#)

**Project:**

Scan Information ([show less](#))

- dependency-check version: 6.0.2
- Report Generated On: Sun, 18 Oct 2020 20:38:39 +0200
- Dependencies Scanned: 35 (37 unique)
- Vulnerable Dependencies: 7
- Vulnerabilities Found: 76
- Vulnerabilities Suppressed: 0
- NVD CVE Checked: 2020-10-18T20:09:51
- NVD CVE Modified: 2020-10-18T20:01:43
- VersionCheckOn: 2020-10-12T11:16:34

**Summary**

Display: [Show Vulnerable Dependencies](#) ([click to show all](#))

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
bsh-2.0b4.jar	cpe:2.3:a:bsh:bsh:2.0.b4:****	org.marcin.bsh:bsh:2.0.b4	HIGH	1	Highest	22
c3p0-0.9.1-pre9.jar	cpe:2.3:a:c3p0:c3p0:0.9.1-pre9:****	org.marcin.c3p0:c3p0:0.9.1-pre9	HIGH	1	Highest	18
commons-httpclient-3.0-rc3.jar	cpe:2.3:a:commons-httpclient:commons-httpclient:3.0-rc3:****	org.marcin.commons-httpclient:commons-httpclient:3.0-rc3	MEDIUM	1	Highest	36
log4j-1.2.12.jar	cpe:2.3:a:log4j:log4j:1.2.12:****	org.marcin.log4j:log4j:1.2.12	CRITICAL	2	Highest	19
mysql-connector-java-5.0.3-bin.jar	cpe:2.3:a:mysql:mysql:5.0.3:****	org.marcin.mysql:mysql-connector-java:5.0.3	HIGH	7	Highest	30
postgresql-8.0-313-jdbc3.jar	cpe:2.3:a:postgresql:postgresql:8.0-313:****	org.marcin.postgresql:postgresql:8.0-313	CRITICAL	30	Low	5
quartz-1.5.1.jar	cpe:2.3:a:quartz:quartz:1.5.1:****	org.marcin.quartz:quartz:1.5.1	CRITICAL	1	Highest	18
servlet-api.jar	cpe:2.3:a:servlet:servlet:3.0:****	org.marcin.servlet:servlet-api:3.0	HIGH	26	Highest	18
xalan.jar	cpe:2.3:a:xalan:xalan:2.7.0:****	org.marcin.xalan:xalan:2.7.0	HIGH	1	Highest	40

Figure 25 Result from OWASP dependency checker

This report states that the following dependencies has a total of 70 vulnerabilities: *Bsh-2.0b4.jar*, *C3p0-0.9.1-pre9.jar*, *Commons-httpclient-3.0-rc3.jar*, *Log4j-1.2.12.jar*, *Mysql-connector-java-5.0.3-bin.jar*, *postgresql-8.0-313-jdbc3.jar*, *quartz-1.5.1.jar*, *servlet-api.jar*, and *xalan.jar*.

Critical vulnerabilities are found in two of the libraries responsible for the database connection in jForum and in one package responsible for logging.

## References

- A3:2017-Sensitive Data Exposure | OWASP. (2020). Retrieved October 20, 2020, from [https://owasp.org/www-project-top-ten/2017/A3\\_2017-Sensitive\\_Data\\_Exposure.html](https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure.html)*
- A9:2017-Using Components with Known Vulnerabilities | OWASP. (2020). [https://owasp.org/www-project-top-ten/2017/A9\\_2017-Using\\_Components\\_with\\_Known\\_Vulnerabilities.html](https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities.html)*
- Biggest Threat to Application Security: SQL Injection Attacks. (2020). Retrieved October 20, 2020, from <https://www.appknox.com/blog/sql-injection-attacks>*
- Broken authentication and session management | Hdiv Documentation. (2020). Retrieved October 13, 2020, from <https://hdivsecurity.com/docs/broken-authentication-session-management/>*
- Consequences of SQL injection. (2013). IT Security Concepts. Retrieved October 20, 2020, from <https://compsecurityconcepts.wordpress.com/tag/consequences-of-sql-injection/>*
- Cross Site Scripting Prevention—OWASP Cheat Sheet Series. (2020). [https://cheatsheetseries.owasp.org/cheatsheets/Cross\\_Site\\_Scripting\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)*
- Cross Site Scripting (XSS) Software Attack | OWASP Foundation. (2020). <https://owasp.org/www-community/attacks/xss/>*
- Cross-site Scripting | What is XSS Attack? | Netsparker. (2019). Retrieved October 20, 2020, from <https://www.netsparker.com/blog/web-security/cross-site-scripting-xss/>*
- Custom Special Character Injection | OWASP. (2020). Retrieved October 20, 2020, from [https://owasp.org/www-community/attacks/Custom\\_Special\\_Character\\_Injection](https://owasp.org/www-community/attacks/Custom_Special_Character_Injection)*

CWE - CWE-776: *Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')* (4.2). (n.d.). Retrieved October 20, 2020, from <https://cwe.mitre.org/data/definitions/776.html>

CWE - CWE-79: *Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')* (4.2). (2020). Retrieved October 20, 2020, from <https://cwe.mitre.org/data/definitions/79.html>

CWE - CWE-80: *Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)* (4.2). (2020). Retrieved October 20, 2020, from <https://cwe.mitre.org/data/definitions/80.html>

CWE - CWE-89: *Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')* (4.2). (2020). Retrieved October 20, 2020, from <https://cwe.mitre.org/data/definitions/89.html>

CWE - CWE-138: *Improper Neutralization of Special Elements* (4.2). (2020). Retrieved October 20, 2020, from <https://cwe.mitre.org/data/definitions/138.html>

CWE - CWE-184: *Incomplete List of Disallowed Inputs* (4.2). (2020). <https://cwe.mitre.org/data/definitions/184.html>

CWE - CWE-284: *Improper Access Control* (4.2). (2020). Retrieved October 20, 2020, from <https://cwe.mitre.org/data/definitions/284.html>

CWE - CWE-326: *Inadequate Encryption Strength* (4.2). (2020). Retrieved October 20, 2020, from <https://cwe.mitre.org/data/definitions/326.html>

CWE - CWE-359: *Exposure of Private Personal Information to an Unauthorized Actor* (4.2). (2020). Retrieved October 20, 2020, from <https://cwe.mitre.org/data/definitions/359.html>

CWE - CWE-601: *URL Redirection to Untrusted Site ('Open Redirect')* (4.2). (2020). Retrieved October 20, 2020, from <https://cwe.mitre.org/data/definitions/601.html>

CWE - CWE-784: *Reliance on Cookies without Validation and Integrity Checking in a Security Decision (4.2)*. (n.d.). Retrieved October 20, 2020, from <https://cwe.mitre.org/data/definitions/784.html>

Dougherty, C. (2009). *Secure Design Patterns*. 118.

*File Upload—OWASP Cheat Sheet Series*. (2020). Retrieved October 13, 2020, from [https://cheatsheetseries.owasp.org/cheatsheets/File\\_Upload\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html)

*Interface javax.servlet.Servlet | Oracle Documentation*. (2020).

Retrieved October 19, 2020, from [https://docs.oracle.com/cd/E17802\\_01/products/products/servlet/2.1/api/javax.servlet.Servlet.html](https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.1/api/javax.servlet.Servlet.html)

*Input Validation—OWASP Cheat Sheet Series*. (2020). [https://cheatsheetseries.owasp.org/cheatsheets/Input\\_Validation\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html)

*OS Command Injection Defense—OWASP Cheat Sheet Series*. (2020). Retrieved October 20, 2020, from [https://cheatsheetseries.owasp.org/cheatsheets/OS\\_Command\\_Injection\\_Defense\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/OS_Command_Injection_Defense_Cheat_Sheet.html)

*OWASP Dependency-Check Project | OWASP*. (2020). Retrieved October 13, 2020, from <https://owasp.org/www-project-dependency-check/>

*OWASP Top Ten Web Application Security Risks | OWASP*. (2020). Retrieved October 20, 2020, from <https://owasp.org/www-project-top-ten/>

Rosado, D. G., Gutierrez, C., Fernandez-Medina, E., & Piattini, M. (2006). A study of security architectural patterns. *First International Conference on Availability, Reliability and Security (ARES'06)*, 8 pp. – 365. <https://doi.org/10.1109/ARES.2006.18>

*Session hijacking attack Software Attack / OWASP Foundation. (2020.). Retrieved*

*October 20, 2020, from [https://owasp.org/www-](https://owasp.org/www-community/attacks/Session_hijacking_attack)*

*[community/attacks/Session\\_hijacking\\_attack](https://owasp.org/www-community/attacks/Session_hijacking_attack)*

*Team, N. S. (2019, April 18). The Cross-site Scripting (XSS) Vulnerability: Definition*

*and Prevention. [https://www.netsparker.com/blog/web-security/cross-site-scripting-](https://www.netsparker.com/blog/web-security/cross-site-scripting-xss/)*

*[xss/](https://www.netsparker.com/blog/web-security/cross-site-scripting-xss/)*

*Unreleased Resource / OWASP. (2020). Retrieved October 20, 2020, from*

*[https://owasp.org/www-community/vulnerabilities/Unreleased\\_Resource](https://owasp.org/www-community/vulnerabilities/Unreleased_Resource)*

*Unrestricted File Upload / OWASP. (2020). [https://owasp.org/www-](https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload)*

*[community/vulnerabilities/Unrestricted\\_File\\_Upload](https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload)*

*What is SQL Injection (SQLi) and How to Prevent Attacks. (2020). Acunetix. Retrieved*

*October 20, 2020, from <https://www.acunetix.com/websitesecurity/sql-injection/>*

*Woschek, M. (2020). OWASP Cheat Sheets. 315.*

# Appendix

## Coverity

```
> "C:\Program Files\Coverity\Coverity Static Analysis\bin\cov-analyze.exe" --dir C:\Group6 -
```

Coverity Static Analysis version 2020.09 on Windows 10 (Unknown Edition number 125), 64-bit  
Internal version numbers: 3412dc3383 p-upland-push-1502

Looking for translation units

|0-----25-----50-----75-----100|

\*\*\*\*\*

[STATUS] Detecting duplicate files

|0-----25-----50-----75-----100|

\*\*\*\*\*

[STATUS] Running framework analysis

|0-----25-----50-----75-----100|

\*\*\*\*\*

[WARNING] The Android Security checkers are enabled by  
--android-security, but no Android applications were captured.  
If present, the Android applications should be captured using  
cov-build filesystem capture, for example with the  
--fs-capture-search or --fs-capture-list options to cov-build.  
To disable this warning use the --skip-android-app-sanity-check  
option.

[STATUS] Resolving dataflow directives

|0-----25-----50-----75-----100|

\*\*\*\*\*

[STATUS] Loading topological sort from disk (11973 functions)

|0-----25-----50-----75-----100|

\*\*\*\*\*

[STATUS] Preparing for bytecode analysis

|0-----25-----50-----75-----100|

\*\*\*\*\*

INFO: 2 source files are generated code, e.g. from JSP, and will be skipped for SpotBugs.

[STATUS] Running SpotBugs analysis

Using SpotBugs

Scanning archives (27 / 27)

2 analysis passes to perform

Pass 1: Analyzing classes (756 / 756) - 100% complete

Pass 2: Analyzing classes (354 / 354) - 100% complete

Done with analysis

2 analysis passes to perform  
Pass 1: Analyzing classes (756 / 756) - 100% complete  
Pass 2: Analyzing classes (354 / 354) - 100% complete  
Done with analysis

SpotBugs time: 00:00:17

```
*****
[STATUS] Preparing for source code analysis
|0-----25-----50-----75-----100|
*****
[STATUS] Running JSHint analysis (jshint v2.9.5)
|0-----25-----50-----75-----100|
*****
[STATUS] Computing node costs
|0-----25-----50-----75-----100|
*****
[STATUS] Running analysis
|0-----25-----50-----75-----100|
*****
[STATUS] Exporting summaries
|0-----25-----50-----75-----100|
*****
[STATUS] Calculating cross-references
|0-----25-----50-----75-----100|
*****
```

Analysis summary report:

```
-----
Files analyzed           : 460 Total
   JSP                   : 3
   Java                   : 335
   Java (without build)  : 26
   JavaScript             : 60
   Text                   : 36
Total LoC input to cov-analyze : 44590
Functions analyzed       : 11257
Classes/structs analyzed : 505
Paths analyzed           : 561511
Time taken by analysis   : 00:02:54
Defect occurrences found  : 114 Total
```

\*\*\*\*\*

# Analysis summary report:

-----

Files analyzed	: 460 Total
JSP	: 3
Java	: 335
Java (without build)	: 26
JavaScript	: 60
Text	: 36
Total LoC input to cov-analyze	: 44590
Functions analyzed	: 11257
Classes/structs analyzed	: 505
Paths analyzed	: 561511
Time taken by analysis	: 00:02:54
Defect occurrences found	: 114 Total
	14 BAD_LOCK_OBJECT
	3 CALL_SUPER
	2 CHECKED_RETURN
	1 CONFIG.MISSING_GLOBAL_EXCEPTION_HANDLER
	1 CSRF
	16 DIVIDE_BY_ZERO
	1 DOM_XSS
	6 FORWARD_NULL
	1 GUARDED_BY_VIOLATION
	2 INSECURE_COMMUNICATION
	2 LOCK_EVASION
	1 MISSING_AUTHZ
	2 NON_STATIC_GUARDING_STATIC
	26 NULL_RETURNS
	2 OVERFLOW_BEFORE_WIDEN
	20 RESOURCE_LEAK
	1 RISKY_CRYPT0
	3 SQLI
	5 SWAPPED_ARGUMENTS
	1 TRUST_BOUNDARY_VIOLATION
	1 UNENCRYPTED_SENSITIVE_DATA
	1 UNLOGGED_SECURITY_EXCEPTION
	1 UNSAFE_REFLECTION
	1 WEAK_PASSWORD_HASH
Additional defects, SpotBugs	: 38
Additional defects, JSHint	: 36