Stockholm University

software security

Autumn 2020

Group 11 Assignment 4
Final Hand-in

Authors:
Helena Lesley
Madeleine Mikaelsson
Hanna Stråle Weber
Felicia Johansson Zetterlund
Qijie Ye

Part 1) Delegation of work

The tasks have been delegated as seen below, all group members have read the provided material.

Name	Delegation
Felicia	WebScarab/Wireshark, Code inspection - Dao, Fortify
Madeleine	Database, Code inspection - Templates, Vulnerabilities/issues without static analysis tools: Database/sensitive data exposure & cross site scripting
Qijie	XSS penetration test, Code inspection - SSO, Templates, JForum Architecture
Helena	WebScarab/Wireshark, Code inspection - Dao, Coverity Static Analysis tool
Hanna	Nmap, Code inspection - Security, Vulnerabilities/issues without static analysis tools: HTTP, unencrypted passwords & MD5

Part 2) JForum Architecture

JForum built an MVC (Model-View-Controller) framework (JForum2, 2020). The architecture generally has three components, model, view, and controller. The model is responsible for managing the data of the application, which contains business logic and information about the types of data in the database (Wayner, n.d.). It receives user input from the controller (Model–view–controller, 2020).
JForum uses Freemarker HTML Templates as a base. The view is responsible for displaying the prepared data from the model by using a template. The templates focus on presentation issues like visual design, layout, and formatting (What is Apache FreeMarker™?, 2020).
The controller connects the database and template. It has various rules and methods for transforming the data moving between view and model. It receives the user input, optionally validates it, and then translates the

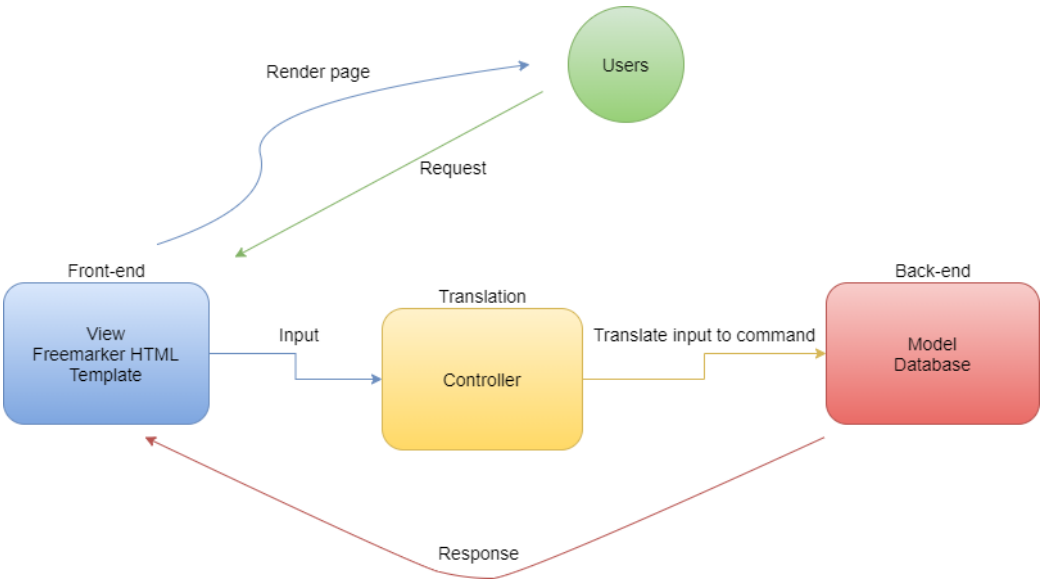


Figure 1: Model over an MVC framework

input to command data model objects (manipulate the database) (Model–view–controller, 2020).

How a request/response is handled, including how URLs translate into actions?
JForum accepts and responds to the Hypertext Transfer Protocol requests. The client opens a connection (mostly by a browser) to the servlet and sends a request. The request contains information including a method like GET or POST, and a URL indicating which resource is being requested (Sun Java System Web Server 6.1 SP11 NSAPI Programmer's Guide), 2010).

According to the code in RequestContext.java in the JForum source code folder, the server receives a request and processes it. Each request will be broken down into a series of steps that together make up the request-handling process ((How the Server Handles Requests from Clients (Sun Java System Web Server 6.1 SP11 NSAPI Programmer's Guide), 2010). These operations include getRequestURL(), getQueryString(), getHeader(), getRemoteAddr() and etc.

How Freemarker HTML templates are used to render page content?

Freemarker HTML templates look similar to regular static HTML, but in order to work effectively with some minor data changes, it contains some instructions which make it dynamic. Whenever someone visits this page, FreeMarker will step in and transform the template on-the-fly to plain HTML by replacing the $...-s with up-to-date content, and send the result to the visitor's Web browser (Manual, Guide, Started and output, n.d.).

How basic application properties are configured?

JForum stored application properties in the jforum-src/WEB-INF/config/SystemGlobals properties file, which contains JForum's general and basic settings about homepage link, default language, page encoding, etc. Database settings show database type to use, the permission of transaction, etc. SSO settings show the authentication type of JForum is non-SSO authentication. Cache settings, general board configurations, and other settings are stored in the properties file.

How the database backend is used?

The database is used to store various information of JForum like users' personal information such as user ID and user passwords, user's activities on JForum such as attachments, posts, pictures, banners, etc. According to the jforum-src/WEB-INF/config/Database folder, JForum supports different database servers, including MySQL, hsqldb, sqlserver, PostgreSQL, and oracle, and corresponding SQL files are saved to manage and manipulate the database.

How access control and sessions are achieved?

Login credentials manage the access control. The authenticator will validate the input of usernames and passwords against the database. According to the jforum-src/src/net/jforum/security/PermissionControl.java file, JForum seemingly has role-based access control. It creates different role names and permission sections within different permission items, and also has a method called CanAccess, which input a string RoleName and return a boolean data type.

According to the jforum-src/src/net/jforum/SessionFacade.java file, JForum verified if there is a user in the session by user_id and session_id. If the user_id exists in the session, then return the session_id. Notably, there is a UserSessionDAO in the DAO folder, and the session id will be persistent in the database.

Part 3) Vulnerabilities and Issues without the use of Static Analysis Tools

3.1 MD5

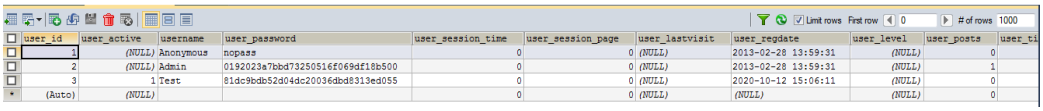
Through manual inspection of the sso file of the source code for JForum, it was found that MD5 is the hashing algorithm used to encrypt passwords. Although there are positive aspects of MD5, like quick processing time, the hashing algorithm suffers from several vulnerabilities that can produce vulnerabilities to the JForum architecture. The MD5 algorithm is, for example, especially vulnerable to collision attacks, which involve two different inputs creating the same hash value. This is something that should not be possible for a cryptographic algorithm to be considered secure. A tool that could be used to exploit this vulnerability in the JForum is Hashcat. Hashcat can perform multiple types of attacks, including brute force and dictionary attacks.

The use of MD5 for hashing in the JForum architecture can affect both the confidentiality and integrity of the system (Libed, Sison & Medina, 2018). This type of weak encryption method is something that can cause exposure of sensitive data, which is one of the OWASP top 10 vulnerabilities (OWASP, 2017). To fix this vulnerability, a stronger hashing algorithm would have to be used. An example of a hashing algorithm that is currently not known to suffer the same vulnerabilities as MD5 is SHA-2 (Martino & Cilardo, 2020).

```
89          p = JForumExecutionContext.getConnection().prepareStatement(
90                  SystemGlobals.getSql("UserModel.login"));
91          p.setString(1, username);
92          p.setString(2, MD5.crypt(password));
```

Figure 2: Code inspection, MD5

The encryption of the passwords was after the code inspection was confirmed by looking at how they were stored in the database. In Figure 3 below, we can see how the user information is stored; this is found in the table “jforum_users.”



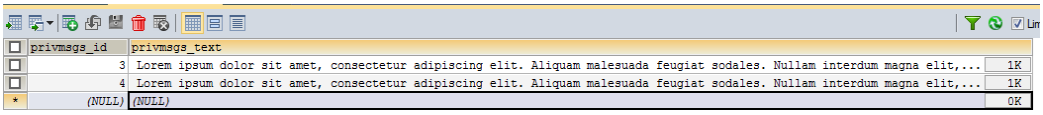
user_id	user_active	username	user_password	user_session_time	user_session_page	user_lastvisit	user_regdate	user_level	user_posts	user_ti
1	(NULL)	Anonymous	nopass	0	0	(NULL)	2013-02-28 13:59:31	(NULL)	0	
2	(NULL)	Admin	0192023a7bbd73250516f069df18b500	0	0	(NULL)	2013-02-28 13:59:31	(NULL)	1	
3	1	Test	81dc9bdb52d04dc20036dbd8313ed055	0	0	(NULL)	2020-10-12 15:06:11	(NULL)	0	
(Auto)	(NULL)			0	0	(NULL)	(NULL)	(NULL)	0	

Figure 3: Database table for user information

3.2 Sensitive Data Exposure

The case of sensitive data exposure was found in the MySQL database. The find was done by manually going through the tables in the database and looking for anything that was thought to be unusual or a possible vulnerability. The data exposure is in regards to private messages sent between users being stored in plain text. This was found under the table "jforum_privmsgs_text," where the columns are "privmsgs_id" and "privmsgs_text," as seen in Figure 4. The picture also demonstrates how the messages, as seen under "privmsgs_text," are stored in plain text.

The storing of the messages in plain text might be an intentional design decision, but in our opinion, this is a case of sensitive data exposure from OWASP Top 10 (OWASP, 2017). Storing the messages in plain text will make it so that anyone having or gaining access to the database might find incriminating information in the messages. Seeing as the messages are stored in plain text, then it can stand to reason that they are transferred in plain text between the application and the database. Making it possible to read the messages as they are being sent in between by someone listening to the connection. To solve this, one can add encryption for the messages stored in the database and also make sure they are encrypted during transfer as well.



privmsgs_id	privmsgs_text	
3	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam malesuada feugiat sodales. Nullam interdum magna elit,...	1K
4	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam malesuada feugiat sodales. Nullam interdum magna elit,...	1K
(NULL)	(NULL)	0K

Figure 4: The table "jforum_privmsgs_text."

3.3 Cross-Site Scripting

Cross-site scripting (XSS) is one of the most common vulnerabilities found in OWASP top 10; the risk of XSS can range from a nuisance to severe se-

curity risk (Conklin and Robinson, 2020). This vulnerability was found by trying to make a reflected XSS attack, also considered non-persistent. The page used for the attack is the login page where the attack was made by inputting a short script of ¡script¿alert(“hack”)¡/script¿ in the username box and some random characters for the password. The alert done by the script immediately pops up on the screen, as seen in Figure 5.

The XSS attack demonstrated previously is at its most basic. This only to illustrate the possibility of a cross-site scripting attack being possible. More ruinous examples of XSS are page tampering, session hijacking, stealing cookies, and disclosing user data (Team, 2019). These attacks can have devastating effects, and to make sure they are not possible, input validation is needed. At its current form, JForum trusts all inputs by the user to be correct. This is both not a realistic mindset but also dangerous. All inputs need to be controlled for the expected length or check for sensitive characters before they are sent further and executed.

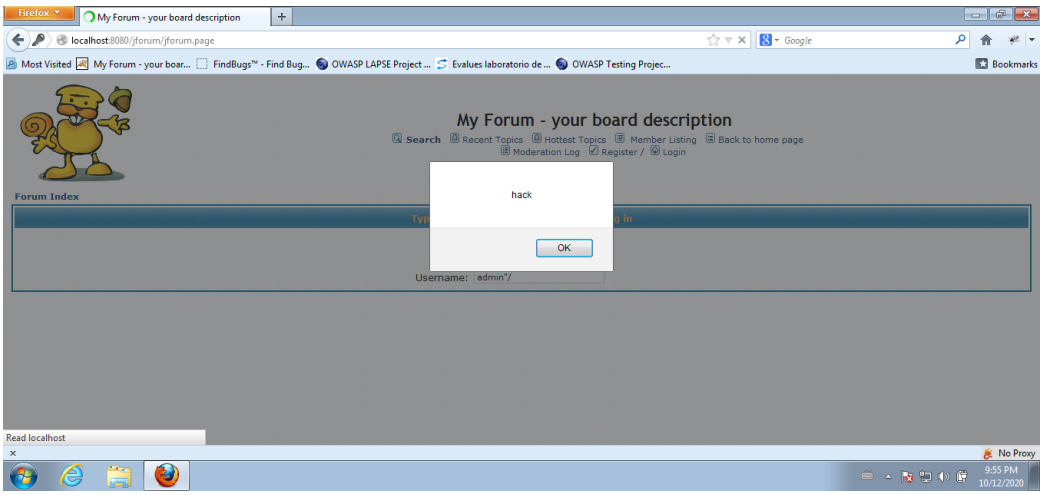


Figure 5: Reflected XSS attack

3.4 Insecure communication

When testing the JForum source code with the tool Wireshark, it showed that the website lacks in use of encryption of transmitted data, the packet traffic is shown in Figure 6. The communication protocol HTTP is not encrypted

using Transport Layer Security (TLS) and this means that information under transit, in HTTP packets, can be exploited by a Man in the Middle attack. In such an attack an attacker is listening to the network that is used and can capture valuable and crucial information that the user of the website does not want the attacker to find out about. The website can be vulnerable to attacks of broken authentication, SQL injection, XSS, and more. It is a crucial security risk as the intruder could gain information that could lead to him/her gaining access to one's account, changing files, and manipulating the website.
All this can be done without the real user, knowing what is going on behind the scene by the hacker. The attack is often known after the attack has been successfully made and thus too late to recover from (Purevpn, n.d.). The tool that was used to test the JForum, Wireshark, is a network analysis tool to capture packets under transit. This same tool could also be used by a malicious attacker to perform a man in the middle attack. To fix this vulnerability, a TLS extension to the HTTP communication protocol would have to be implemented so that the communication will be encrypted.

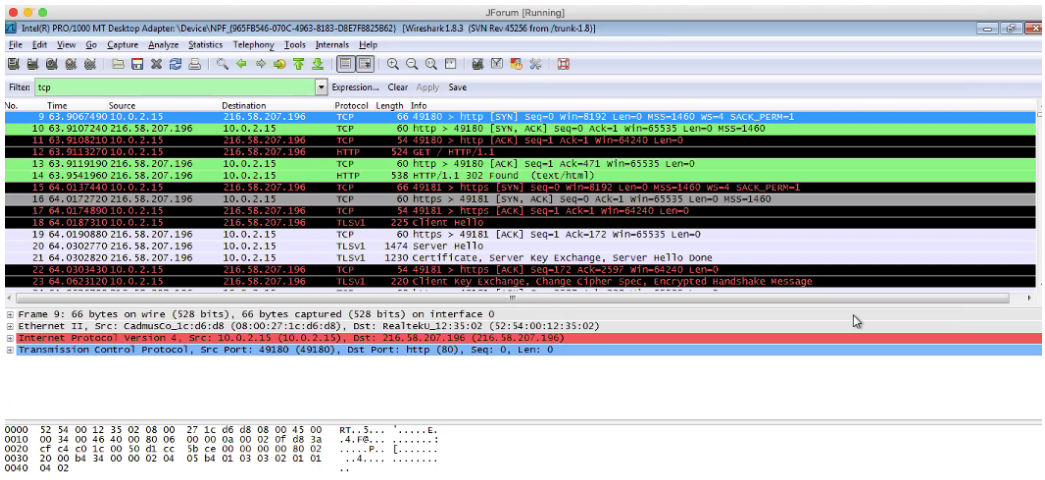


Figure 6: Wireshark, Packet traffic

3.5 Unencrypted Authentication Credentials

The group used WebScarab, a testing tool that was found in the VM, to see if it was possible to capture usernames and passwords in transit. The

connection was set up using the default proxy in WebScarab and then the proxy in the browser was changed to the same one. After that, the JForum website was visited and the admin credentials were used to log in. Because requests were intercepted in WebScarab, Figure 7 popped up. This is the request sent from the browser to log in to JForum. At the bottom line, the username and password are both shown in cleartext.

In the first attempt, the login credentials provided to the group in iLearn

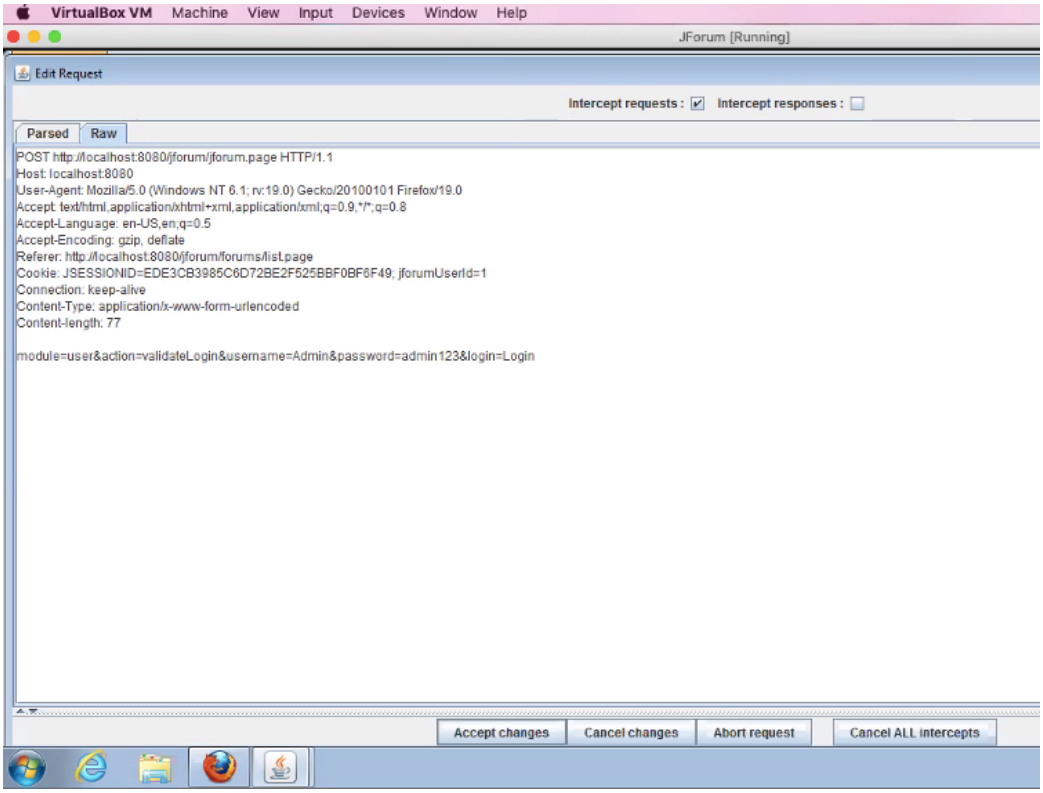


Figure 7: Request window in WebScarab

were used. In a second attempt, made up credentials were used instead (pressing random keys on the keyboard) to see what would happen then. The request looked the same, with username and password in cleartext. After that, the captured packet was transferred to the Fuzzer tab in WebScarab, Figure 8, and here the username and password were once again shown in cleartext. This proves that WebScarab is a tool that could potentially be

used for an attack on the JForum.

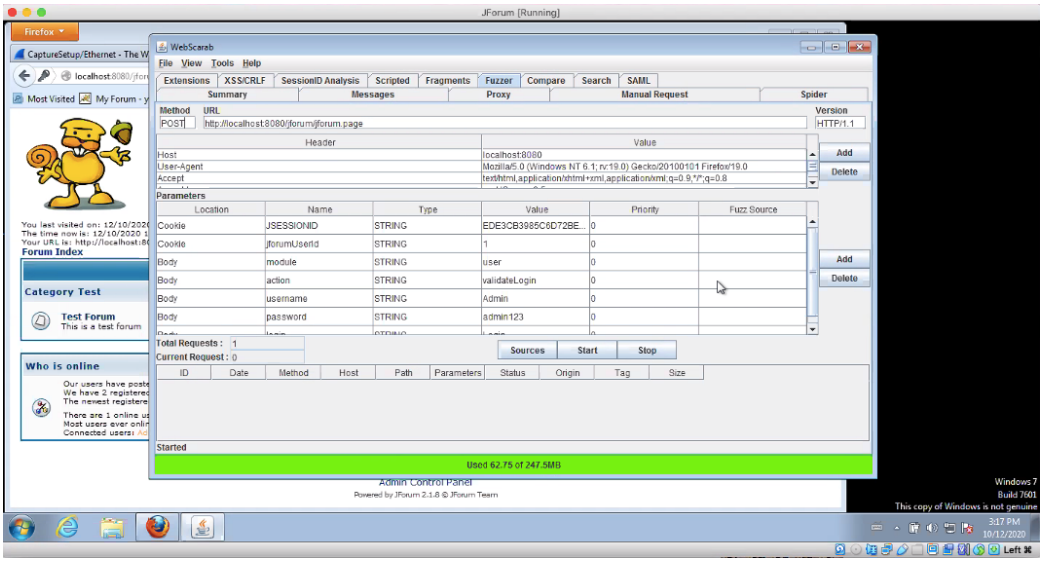


Figure 8: Webscarab, fuzzer tab

Sensitive data exposure is one of the vulnerabilities that can be found in the OWASP Top 10. Sensitive data exposure can happen in several ways, but one of them is stealing data in transit when sent in cleartext. To prevent this, all sensitive data in transit should be encrypted so that information, like usernames and passwords, will not be shown in cleartext (OWASP, 2017).

Part 4) Use of Static Analysis Tools

Coverity

Coverity Static Analysis tool is a program used to find vulnerabilities while programming or checking for vulnerabilities in software programming code. It shows an easily adaptable and clear description of the vulnerabilities that have been found and does it continually throughout the whole workflow. After installing Coverity according to the downloading descriptions, we added the JForum folder, with JForum Java code, to be run through Coverity Analysis. After checking the Java code, we looked at our desktop, entered the

JForum map, and looked for the output folder. In the output folder, we found summary.txt, which showed us the below vulnerabilities in the JForum Java code.

The file Summary.txt showed that we received 114 vulnerabilities that had been found through the Coverity analysis run of JForum Java code. The defect occurrences found are the different names of vulnerabilities and where we can find a more in-depth description of each vulnerability in our JForum output folder. It also tells us how many of each vulnerability description Coverity has found. Below headlines present what vulnerability we received.

No Salt - MD5 and Risky Crypto
In the files of WEAK_PASSWORD_HASH and RISKY_CRYPTO, we saw that the JForum code didn't apply salt to complement the hash algorithm of MD5. That is seen as a vulnerability due to not creating unique password hashing strings for all JForum users. Salt is a safeguard of passwords in storage (Wikipedia Contributors, 2019).

XSS
In the file of DOM_XSS, we saw that Jforum had vulnerabilities of HTML injection, JavaScript execution, and manipulation of the URL string. According to XSS, different text or code injection in different search fields, URL browser or login and passwords fields at Jforum webpage, could cause tampering page information, session hijacking, stealing cookies, disclosing user data, etc. (Team, 2019).

SQL Injection
SQLI file showed vulnerabilities of SQL and My BilindSQL injection at the JForum webpage. JForum database saves user email and password in database folders named jforum_users, which indicate that it may be easy for hackers to guess where to find all users' email and passwords in an SQL attack. To solve this problem, the database folder of jforum_users could be named something else, making it harder for hackers to guess where to find them.

Unencrypted Sensitive Data
The file of UNENCYPTED_SENSITIVE_DATA showed that user email and password files in the database of JForum are not encrypted. That means that SQL injection could be a successful attack. If the files were encrypted,

the hacker could not be able to view the information of such files. The solution to this is to encrypt all data viewed as sensitive information in the JForum database.

URL Vulnerability
INSECURE_COMMUNICATION file showed vulnerabilities of the URL browser of the JForum webpage. The URL of the JForum webpage is not encrypted, due to the use of HTTP. It can be seen as vulnerable for hackers listening, interception, or eavesdropping on the transmitted data on the webpage. In the transmitted data, the hacker could gain valuable and sensitive information. The solution to this would be to apply SSL and by so the usage of HTTPS. HTTPS encrypts all transmitted data during transit (globalsign, n.d.)

General reflections of Coverity usage
Coverity Static Analysis tool was in some bit hard to install, due to the time to download it and error shown in Test Build of the configuration of the program and applying JForum to be run in the program. However, when Coverity had analyzed the JForum code, it was easy to find what vulnerabilities the program found in the output summary file. In the summary file, it was user friendly for a novice user to see how many vulnerabilities JForum had, as well as where to find more information and description of each vulnerability. In the output folder, each vulnerability in the summary file was named after the summary file description, and that made it easier for us to find the exact file that we wanted to look more into. In each vulnerability file, it was a well-explained description of what the vulnerability was, where it was found in the code, and how it could be solved. Coverity could be seen as a good program for people that aren't used to knowing what different vulnerabilities means and how it can be solved. Coverity helps the user of the program as a guide in what to look for, what the vulnerabilities are found, and the solution to it. It was clear that Coverity found things that we didn't know about from before. By manually checking we say few vulnerabilities but with Coverity, we received a lot more.

Fortify

Fortify is a static code analyzer that can be used to analyze the source code of the software. This is the sourceanalyzer tool in Fortify. To be able to use

it, we had to connect to DSV:s Linux server triton. After the connection had been established, we ran the suggested three lines of code. After that, we used sourceanalyzer from the Terminal window and searched for any other java files we wanted to look at. We decided that we would look in the folders and files we already had prioritized in previous hand-ins.

The scannings did not provide any new insights as to what vulnerabilities or weaknesses that can be found in the JForum code. It did, on the other hand, confirm things that we had also found out in other ways. For example, MD5 is a weak encryption algorithm, and that password management is weak and, therefore, susceptible to sensitive data exposure. Some of the results from the scans can be seen in the appendix, under the Fortify section. We did look at other files as well but did not find anything specific that pointed towards any vulnerabilities.

Using Fortify in this way requires you to manually type in the java files you want to be analyzed, this means in theory that you can check all the java files in the JForum source code, but in practice, this would require more time than we have. Prioritizing what files to look at was, therefore, necessary. Thus, vulnerabilities may have been missed since inexperienced users might not know where the best places in the code to look are. Even though the tool itself was easy to use and the input you provided was quite minimal (i.e. does not require that much from the user), it would still be more beneficial to know what you were looking for and especially where you can look for it.

Reference

Apache FreeMarker™. 2020. What Is Apache Freemarker™?. [online] Available at https://freemarker.apache.org/ [Accessed 6 October 2020].

Conklin, L., and Robinson, G. (2017). CODE REVIEW GUIDE RELEASE Creative Commons (CC) Attribution. [online] Available at: https://owasp.org/www-pdf-archive/OWASP_Code_Review_Guide_v2

Coverity, I. (2014). The Coverity Platform - From a Developer's Perspective. YouTube. Available at: https://www.youtube.com/watch?v=_Vt4niZfNeA [Accessed 19 Oct. 2020].

Docs.oracle.com. 2010. How The Server Handles Requests From Clients (Sun Java System Web Server 6.1 SP11 NSAPI Programmer's Guide). [online] Available at: https://docs.oracle.com/cd/E19857-01/820-7655/abvah/index.html[Accessed 20 October 2020].

En.wikipedia.org. 2020. Model–View–Controller. [online] Available at: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller [Accessed 6 October 2020].

Libed, J. M. Sison, A. M. & Medina, R. P. (2018). Improved MD5 through the extension of 1024 Message Input Block. ACM International Conference Proceeding Series: New York Martino, R. & Cilardo, A. (2020). SHA-2 Acceleration Meeting the Needs of Emerging Applications: A Comparative Survey. IEEE Access.

Manual, A., Guide, T., Started, G. and output, T., n.d. Template + Data-Model = Output. [online] Apache FreeMarker Manual. Available at: https://freemarker.apache.org/docs/dgui_quickstart_basics.html [Accessed 20 October 2020].

OWASP (2017). A3:2017-Sensitive Data Exposure — OWASP. [online] Available at: https://owasp.org/www-project-top-ten/2017/A3_2017 Sensitive_Data_Exposure.html [Accessed 19 Oct. 2020].

www.purevpn.com. (n.d.). What is HTTP Vulnerability & Its Types. [online] Available at:
https://www.purevpn.com/ddos/http-vulnerability [Accessed 13 Oct. 2020].

SourceForge. 2020. Jforum2. [online] Available at:
https://sourceforge.net/projects/jforum2/ [Accessed 5 October 2020].

Team, N., 2019. The Cross-Site Scripting (XSS) Vulnerability: Definition And Prevention. [online] Netsparker.com. Available at:
¡https://www.netsparker.com/blog/web-security/cross-site-scripting-xss/¿ [Accessed 19 October 2020].

Wikipedia Contributors (2019). Salt (cryptography). [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Salt_(cryptography).

www.globalsign.com. (n.d.). What's the difference between HTTP and HTTPS? [online] Available at:
https://www.globalsign.com/en/blog/the-difference-between-http-and-https.

Wayner, P., n.d. How To Choose The Right Software Architecture: The Top 5 Patterns. [online] TechBeacon. Available at:
https://techbeacon.com/app-dev-testing/top-5-software-architecture-patterns-how-make-right-choice [Accessed 6 October 2020].

Appendix

Coverity

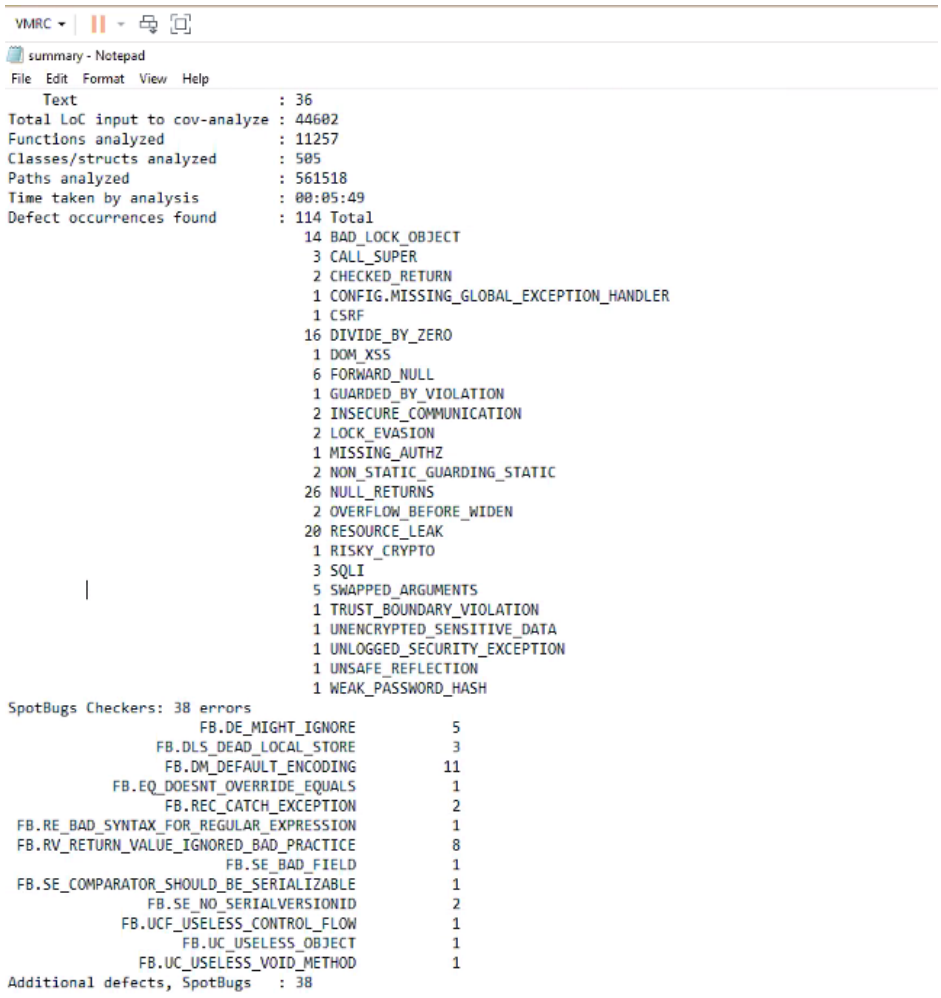


Figure 9: Coverity Summary.txt

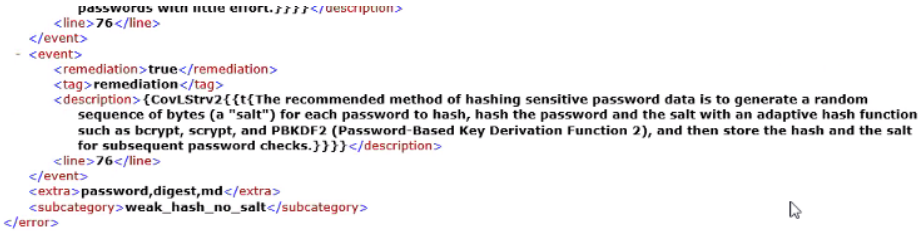


Figure 10: No Salt and MD5

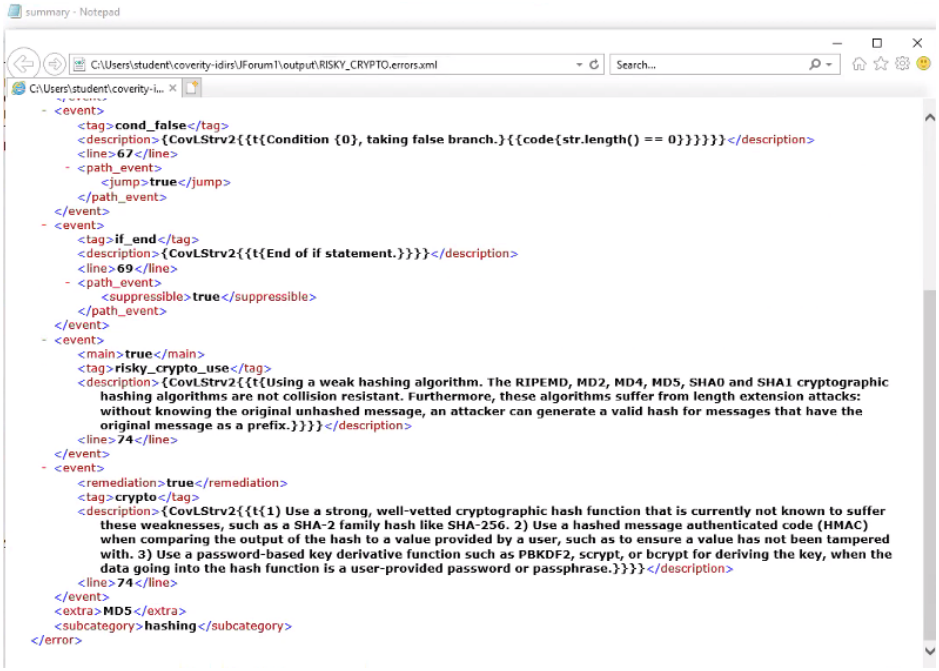


Figure 11: RISKY_CRYPTO

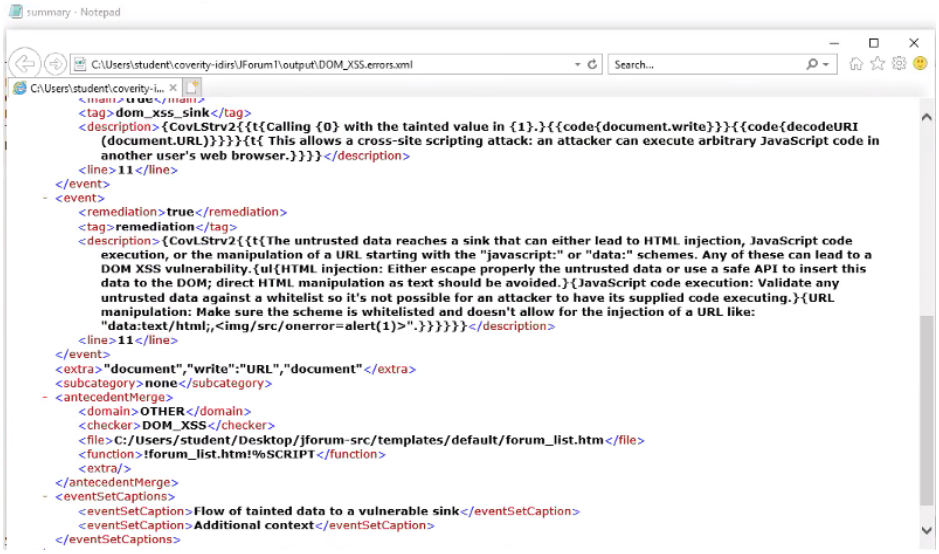


Figure 12: DOM_XSS

Figure 13: SQLI

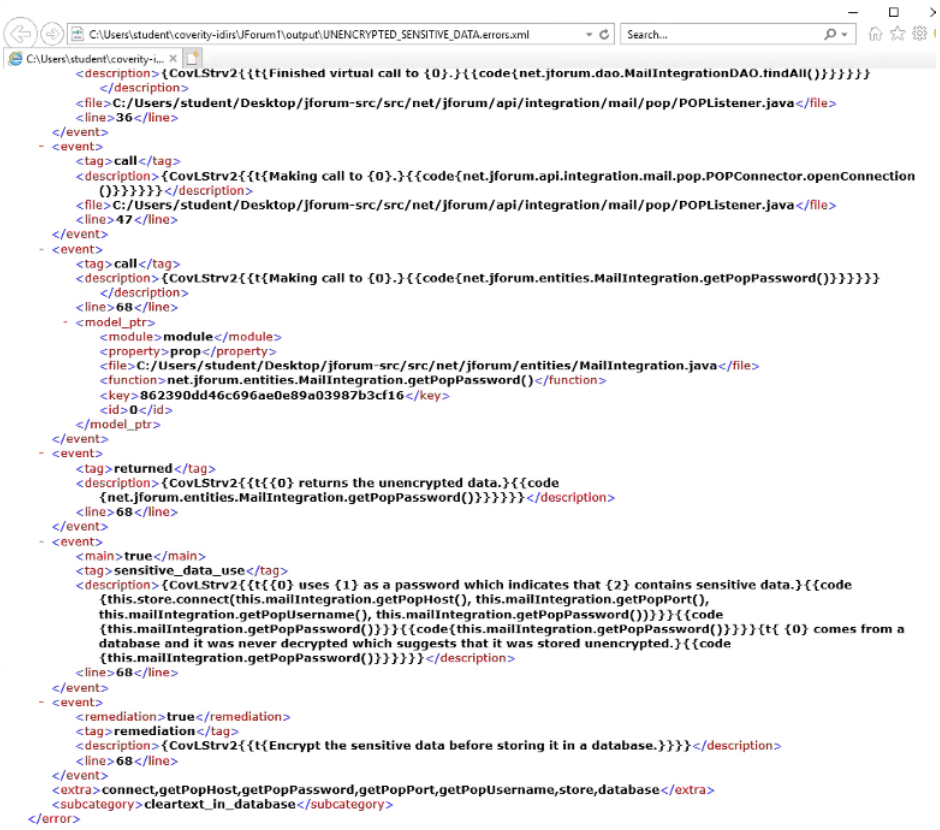


Figure 14: UNENCRYPTED_SENSITIVE_DATA

Figure 15: UNSECURE_COMMUNICATION

Fortify

Figure 16: First scan

```
[/opt/sosec-source/jforum-src/src/net/jforum/util]

[5FF8D076554FCA2A1BF1635F188F4DA3 : low : Weak Cryptographic Hash : semantic ]
MD5.java(74) : MessageDigest.getInstance()
```

Figure 17: MD5

```
[/opt/sosec-source/jforum-src/src/net/jforum/dao]

[49FDDD779FB49A1C9DB7FB7BFB2CA5E8 : low : Password Management : Password in Comment : structural ]
    UserDAO.java(261)

[1AED54237C2A69DE8A9816AFE54AE609 : low : Password Management : Password in Comment : structural ]
    UserDAO.java(286)

[12054BA82F0F0C6495649AB9F5EA965B : low : Password Management : Password in Comment : structural ]
    UserDAO.java(305)
```

Figure 18: User DAO