Software Quality Engineering

LOG8371E

TP3

Nov 30, 2022

The Groupers

Philipp Peron	**Victor Mesterton**	**Rui Jie Li**
philipp.peron@polymtl.ca	victor.mesterton@polymtl.ca	rui-jie.li@polymtl.ca

Rad Ali	**Cedric Helewaut**
rad.ali@polymtl.ca	cedric.helewaut@polymtl.ca

Table of Contents

Abstract

In this third assignment report, we describe our methods and results in analysing the Mango software's security. We performed a static analysis of the source code of the entire Mango application. For this, we used SonarCloud in conjunction with SonnarScanner, and we saw that most aspects of the code had the worst possible score, with the exception of maintainability. We further analysed the system's security features with a penetration test using the OWASP ZAP tool. This resulted in many security alerts ranging from high, medium and low risk. A comparison of both these methods explains the different results obtained and proves that neither one of these methods is objectively better than the other, with each having their advantages and disadvantages.

Introduction

Nowadays, sensors are everywhere. The gathering and interpretation of the sensors' data is therefore crucial. Mango is a software to store such sensor data and create alarms for certain events. Because Mango is mostly used in the commercial sector, its quality is especially important and problems can lead to large revenue loss. It is therefore essential to make sure that it functions correctly and performs as expected in a secure manner. For assignment one, we selected Mango's SQL feature, which provides access to the system's SQL-based database. This allows clients to query and analyse the data of their sensor based systems. In the previous assignment, we focused on performance efficiency, which describes the performance relative to the amount of resources used under stated conditions [9]. Contrary to the two previous assignments, we did not focus on a specific feature of the Mango system. Instead, the tests and analysis we ran were aimed at the system as a whole. This approach allowed us to obtain the highest number of alerts to analyse.

The rest of the report is structured as follows: In section 1, we present the static analysis of the entire Mango application, with eight vulnerabilities or security hotspots from at least three different types. Section 2 describes the penetration test that was applied to the Mango system, with active scans and webcrawlers. The comparison of the results obtained in sections 1 and 2 are described in section 3. This is followed by the conclusion.

1 Static Analysis

We performed a static analysis of the entire Mango application to identify security issues. The static analysis was done using the SonarCloud software. Static analysis means that it is performed on the source code without having to access or enter the application. We will discuss the obtained results in the following sections.

1.1 Summary of the Results

The summary window in figure 1 indicates that 168 thousand lines of code were analysed during the static analysis. From this analysis SonarCloud found a total of 3600 bugs of 7 different types. Indicating that there are multiple bugs of the same type. The Issues are also divided into 3 different severities: blocker, critical and major issues. Additionally, 2 vulnerabilities were found of 1 type. These add to a total of 8 types of issues/vulnerabilities in the mango application that we are going to present and discuss further in section 1.2. Furthermore, we have a total of 4500 code smells present in the project. Code smells are anomalies on the code level that point to bad development practices. They can include duplicate code, code changes required in multiple places, and breaking of the SOLID principles. Therefore, this affects the various quality attributes of a system, such as testability, maintainability, understandability and complexity. Lastly, the mango system has 47 security hotspots and duplication of 25.4%.

To conclude, the Mango system has a lot of issues and has very bad scores in quality attributes. This can be seen from both the number of bugs and code smell and also from the ratings given by SonarCloud. The ratings are the colored circles at the right side of the boxes with a letter inside them, or the colored circle outlining the duplications. They range from A to E, where A is good and E is bad. 3/4 quality attributes are ranked the worst, with the exception of maintainability with an A. Regarding duplication, the worst score is given if it is >20% and a good score would be < 3%, indicating that Mango obtained a very bad score.

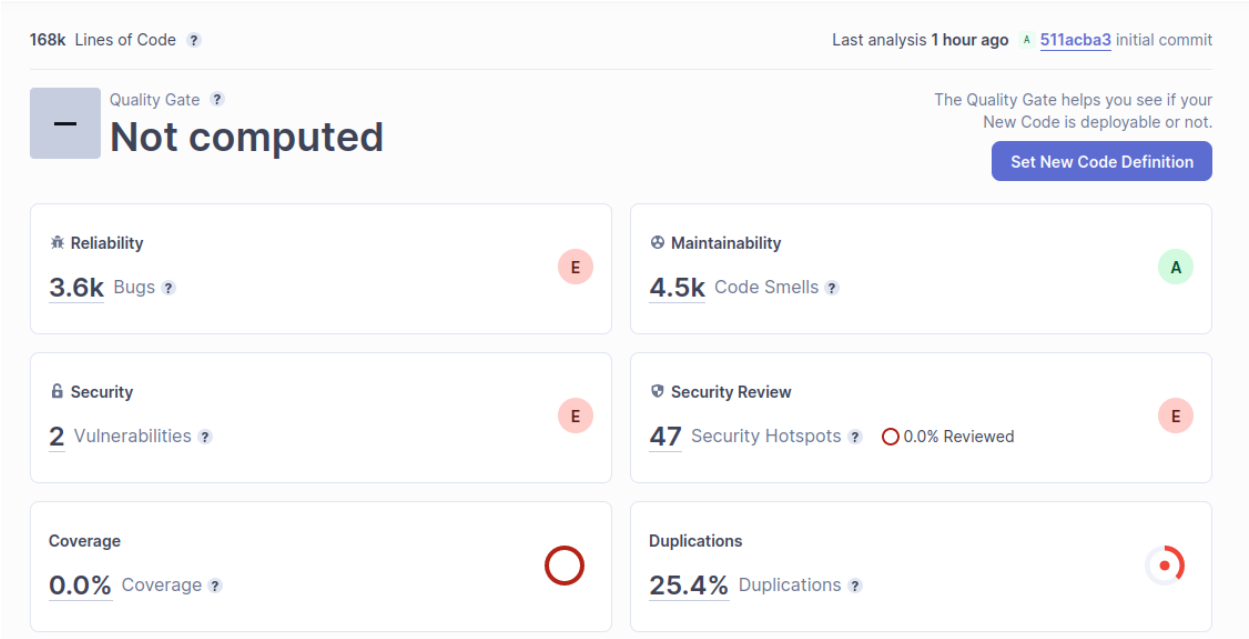

Figure 1. Screenshot of the summary window from SonarCloud

1.2 Eight Security Vulnerabilities

We are presenting and discussing 8 vulnerabilities of the Mango application that was provided to us by the SonarCloud application. As there were only 2 vulnerabilities presented, we expanded our discussion to bugs that were found. Nonetheless, we discuss the following aspects regarding every issue/ vulnerability:

- Short description of the vulnerability
- Potential risk
- The file of the vulnerability
- The severity level
- The type of the vulnerability according to OWASP, SANS, or CWE
- Recommendation for solving the problem

1.2.1 Cross-Site Scripting Vulnerability

Description: The code reflects user-controlled data. Endpoints should not be vulnerable to reflected cross-site scripting (XSS) attacks.

Risk: User-provided data, such as URL parameters, POST data payloads, or cookies, should always be considered untrusted and tainted. Furthermore, when processing an

HTTP request, a web server may copy user-provided data into the body of the HTTP response that is sent back to the user. This behavior is called a "reflection". Endpoints reflecting tainted data could allow attackers to inject code that would eventually be executed in the user's browser. This could enable a wide range of serious attacks like accessing/modifying sensitive information or impersonating other users.

File: build/resources/dojo/tests/widget/showPost.php

Severity: Blocker vulnerability

Type of the vulnerability: Injection, Cross-Site Scripting (XSS)

Solution: Use a whitelist to evaluate if the user-provided data is allowed.

1.2.2 Double-Checked Locking

Description: A dangerous instance of double-checked locking is present in the code. Double-checked locking should not be used.

Risk: Using double-checked locking for the lazy initialization of any other type of primitive or mutable object risks a second thread using an uninitialized or partially initialized member while the first thread is still creating it, and crashing the program.

File: src/com/serotonin/mango/Common.java

Severity: Blocker bug

Type of vulnerability: Security misconfiguration

Solution: Do not use double checked locking and synchronise the whole method instead. If however, you want to avoid this, you can also use an inner static class to hold the reference instead, they guarantee lazy loading.

1.2.3 Nested <script> Elements

Description: Nested <script>..</script> elements are present in the code. When parsing a script node, the browser treats its contents as plain text, and immediately finishes parsing when it finds the first closing </script> character sequence. As a consequence, nested script nodes are not possible, because all opening <script> tags found along the way are ignored.

Risk: Web browsers don't support nested <script>...</script> elements. But there is no error in such a case and browsers just close the first encountered <script> tag as soon as

a closing </script> tag is found along the way. So there is a big chance to display something totally unexpected to the end-users.

File: build/resources/dojo/src/io/xip_client.html

Severity: Major bug

Type of vulnerability: Security misconfiguration

Solution: Avoid using nested <script>…</script> elements in code

1.2.4 Suppressing of Propagation

Description: A jump-statement “throw” is used within the scope of a “finally”-block. This can suppress the propagation of any unhandled Throwable which was thrown in the try or catch block.

Risk: By suppressing the propagation of unhandled Throwables, developers could miss important issues with their code. Potentially introducing vulnerabilities. It does not seem that an attack can directly target this issue.

File: src/…/mango/rt/dataSource/sql/SqlDataSourceRT.java

Severity: Critical bug

Type of vulnerability: Security misconfiguration

Solution: Avoid using jump-statements in “finally”-blocks.

1.2.5 Missing Exception Throwing

Description: The variable “et”, which is nullable, does not throw a “NullPointerException”. A reference to null should never be dereferenced/accessed. Doing so should cause a NullPointerException to be thrown.

Risk: This could expose debugging information that would be useful to an attacker, or it could allow an attacker to bypass security measures.

File: src/…/serotonin/mango/rt/event/type/AuditEventType.java

Severity: Major bug

Type of vulnerability: Sensitive Data Exposure

Solution: Make sure all variables that are nullable throw a “NullPointerException”

1.2.6 Missing Header

Description: Tables in the .jsp files do not have proper headers. We should add “<th>” to each “<table>”.

Risk: Assistive technologies, such as screen readers, use <th> headers to provide some context when users navigate a table. Without it the user gets rapidly lost in the flow of data.

File: build/WEB-INF/jsp/publisherEdit/editPersistent.jsp

Severity: Major bug

Type of vulnerability: Security misconfiguration

Solution: Add the headers to the tables.

1.2.7 Missing Declaration

Description: The HTML-file does not contain a <!DOCTYPE> declaration before a <html> tag. This <!DOCTYPE> declaration tells the web browser which version of HTML is being used on the page, and therefore how to interpret the various elements. Validators also rely on it to know which rules to enforce.

Risk: In applications that process XML inputs, attackers could supply XML Files with specially crafted <!DOCTYPE> definitions to perform XML External Entity (XXE) attacks including denial of service attacks and even remote code execution. We call this XML External Entity (XXE) attacks. If you declare the <!DOCTYPE> just before your <html> tag, your html-file will not be impacted by these malicious XML files.

File: build/resources/dojo/demos/gfx/circles.html

Severity: Major bug

Type of vulnerability: Security misconfiguration

Solution: Insert a <!DOCTYPE> declaration to before this <html> tag

1.2.8 Possibility of Deadlocks

Description: The code synchronizes on instances of value-based classes. Since these objects are pooled and potentially reused, you could introduce deadlocks

Risk: The service could shut down and deadlocks can be very hard to locate.

File: src/…/mango/rt/dataSource/meta/MetaDataSourceRT.java

Severity: Major bug

Type of vulnerability: Security misconfiguration

Solution: You can synchronize on a new “Object” instead

1.3 Setup Manual

To perform the Static analysis we used the SonarCloud Software. It performs the analysis statically on the source code, without having to run it. The following steps will explain how to set up and perform the test.

1. Create a new public repository in your github, or fork an existing public repository to your own account.
2. Go to https://sonarcloud.io
    a. Press log in
    b. Press github
    c. Finish connecting your github
3. Once logged in press myProjects
4. Press Analyze new project, if nor visible press + in upper right corner
5. Follow the instructions provided py SonarCloud to finish setup
    a. Set up organization
    b. Choose free plan
    c. Import desired repository (new one previously created)
    d. Select Manually
    e. Select Other (for JS, TS, Go, Python, PHP, …)
    f. Select your operating system
    g. Download folder and extract it
    h. Add bin directory to path
    i. Add new environmental variable SONAR_TOKEN
6. Clone the Git repository to your computer
7. Navigate to the root directory of the project

8. Execute the following command:

```
sonar-scanner.bat \
      -D"sonar.organization=<USERNAME>" \
      -D"sonar.projectKey=<PROJECT_KEY" \
      -D"sonar.sources=." \
      -D"sonar.java.binaries=src/" \
      -D"sonar.host.url=https://sonarcloud.io"
```

Go to SonarQube to read the results by opening the project.

1.3.1 Encountered problems

The setup was quite straightforward by simply following the instructions. There were a few issues present.

The selection of the analysis method was not presented, this was resolved with trial and error until we found the correct one.

The initial command to run the analysis provided by SonarCloud did not work, as it had some issues with java files. To solve this issue we had to add an additional command:

```
-D"sonar.java.binaries=src/" \
```

The program just presented us with 2 vulnerabilities, of one type. This did not fulfil our requirements of presenting and discussing 8 vulnerabilities. Thus, we expanded by including Bugs in the report.

2 Penetration Testing

2.1 Summary of Results

We ran “Active Scan” in OWASP Zap on the entire Mango website. Additionally, with the automated web crawler, Spider, we manually walked through some parts of the website like the SQL-section. A total of 29 alerts were found; for alerts that are directly related to mango, they can be separated into 3 high risk alerts, 5 medium risk alerts, 2 low risk alerts and 5 informational alerts. The rest of the alerts are from google links such as

Google Search and google APIs.

2.2 Comments on Eight Alerts (all descriptions are from OWASP Zap)

2.2.1 Cross Site Scripting (Reflected)

Risk level: high
Category from OWASP top 10: A03:2021 – Injection
URLs:
http://localhost:8080/test/login.htm
http://localhost:8080/test/dwr/interface/%3Cimg%20src=x%20onerror=prompt()%3E

Description: This is a technique that allows the attacker to insert code into a user's browser. The code can be written in any browser supported languages, but is mostly written in HTML and JavaScript. This allows the attacker to control the user's browser and access sensitive data [1].
An example of this attack (from [2]) would be, for example, suppose a web page http://localhost:8080/mypage uses a parameter `name` to display the username of the user (such as http://localhost:8080/mypage?name=username), we can inject code by using a URL like http://localhost:8080/mypage?name=<script>alert(1)</script>, where everything between <script> and </script> will be executed as code. This can be used to steal the user's data.
The potential vulnerability we found in Mango is a non-persistent attack, which is inserted using the URL such as `password=<script>alert(1)</script>`. Another example found was the response from a GET request: `No class by name: <img src=x onerror=prompt()>`. We were not able to execute any code by using either the username or the password parameter, which is to be expected since they are parameters for inputs so they are not inserted as parts of the DOM.
Solution: A solution proposed by OWASP is input validation [1]. Other solutions include:
1. Use an allow list or a block list (e.g. block all inputs that contain valid HTML or javascript code) [1]
2. Using an external library such as Apache Wicket to prevent this sort of attack [1]
3. Using proper encoding for characters that are not alphanumeric [1]

2.2.2 Remote OS Command Injection

Risk level: high
Category from OWASP top 10: A03:2021 – Injection
URL: http://localhost:8080/test/dwr/call/plaincall/MiscDwr.doLongPoll.dwr
Description: This is an attack that takes advantage of the user input to inject commands

(for example, bash in Linux or PowerShell in windows) that are executed with whatever privilege the backend server has [1] (for example, a function that takes user inputs and runs them as bash scripts). The command can be injected using cookies, HTTP requests, etc [1].
For example (taken from [3]), suppose a backend application written in C++ takes the name of a file as argument and executes `cat filename` with the command `system(command)` to display the content of the file. If the string `filename;rm -rf /` is passed as the file name, the command executed by `system` will be `cat filename;rm -rf /` which will not only display the content of the file, but also remove certain files.
When we scanned the Mango application with active scan, OWASP was able to control the response time by sending the command `1&timeout /T 15`, which is a valid command in Windows PowerShell that tells the process to timeout for 15 seconds.

Solution: OWASP proposes multiple solutions to this problem, one of them being to avoid using the command prompt by replacing it with library calls. We can also restrict the application's access to the operating system. If commands must be executed, there needs to be some kind of input filtering, for example by forbidding all characters that are not alphanumeric. Limiting the output can also prevent sensitive data from being returned to the attacker [1].

2.2.3 SQL Injection - Hypersonic SQL - Time Based

Risk level: high
Category from OWASP top 10: A03:2021 – Injection
URL: http://localhost:8080/test/dwr/call/plaincall/MiscDwr.doLongPoll.dwr
Description: In a time-based SQL injection, the attacker sends a SQL query which forces the database to wait for a specified amount of time before responding. Depending on the response time/delay, the attacker can infer if the result of the query is true or false. This attack is relatively slow because an attacker has to enumerate a database character by character to retrieve strings [4].
For example, by injecting a command such as SLEEP, the attacker can detect which parameter is vulnerable (if the query is delayed, we can conclude that the SLEEP command was really executed) [5].
In out case, a POST request to this URL normally takes an average of 60 178, but OWASP was able to make a request take 148 532 ms by sending a request with parameters `field: [httpSessionId], value ["java.lang.Thread.sleep"(15000)]`.

Solution: Part of the solution OWASP proposes is to have input validation on the backend. Since we do not know what kind of database MangoDB uses (JDBC or ASP), we

cannot really use their recommendations on these. However, one solution we can implement with relative ease is to avoid executing commands by concatenating strings or by using commands such as EXEC or EXEC IMMEDIATE [1].
We can also change the application so that the user executing the SQL queries is not admin [1], since as it is, the only user able to use the SQL feature seems to be users with admin rights due to the function `ensureAdmin()`.

2.2.4 .htaccess Information Leak

Risk level: medium
Category from OWASP top 10: A03:2021 – Injection
URL: http://localhost:8080/test/dwr/call/plaincall/.htaccess
Description: .htaccess files are used in Apache HTTP servers to configure a directory without changing the main configuration file. It allows functionalities such as redirects and CORS [6]. If an .htaccess file is compromised, an attacker can alter which features are available [1].

Solution: Access to the .htaccess file could be blocked with commands such as `RewriteCond. [7]`

2.2.5 Absence of Anti-CSRF Tokens

Risk level: medium
Category from OWASP top 10: A01:2021 - Broken Access Control
URL:
http://localhost:8080/test/login.htm
Description: In a cross-site-request-forgery attack, the attacker forces the victim, without their knowledge, to send an HTML request to a target destination. This means that the attacker can perform actions as the victim. The website trusts the user and potentially grants the user special privileges which are exploited by this attack. This vulnerability in particular means that the HTML submissions form does not include anti-CSRF tokens. The tokens are used to verify that an HTML request was sent by the original user and not a third party [1].
When we used an active scan on Mango, OWASP detected that a GET request does not have anti-CSRF tokens.

Solution: Depending on the phase of the application development, OWASP suggests different solutions:
1. During the Architecture and Design phase, one possible solution is to use a vetted library or a framework which prevents this weakness through its design/architecture or

by other constructs like anti-CSRF packages [1].
2. Another possible solution, which can be added during the implementation phase, is to ensure the absence of cross-site scripting issues. These issues can be used to easily bypass most CSRF-defences [1].
3. A third solution to this vulnerability is the addition of unique nonces for each form. Through the verification of these nonces the original user can be verified [1].

2.2.6 Content Security Policy (CSP) Header Not Set

Risk level: medium
Category from OWASP top 10: None; OWASP considers it to be a Protection Mechanism Failure
URL:
http://localhost:8080/test/login.htm
http://localhost:8080/test/watch_list.shtm
Description: Certain types of attacks can be detected and removed through the content security policy (CSP) layer. Suppressed attacks can include cross-site-scripting and data injection attacks. Website providers can use a set of standard HTTP headers which are provided by CSP to declare approved content sources for the website. These sources can include types like JavaScript, CSS, HTMLS frames, images, fonts, etc. By not setting the CSP headers, the attackers have a wider range of possible attacks since they are less restricted in which types of scripts/attacks they can run.[1]

Solution: According to OWASP, the solution to the Content Security Policy (CSP) Header Not Set vulnerability is the configuration of the CSP-header with the respective protocols used by the website [1].

2.2.7 Missing Anti-clickjacking Header

Risk level: medium
Category from OWASP top 10: A04:2021 - Insecure Design
URL:
http://localhost:8080/test/login.htm
http://localhost:8080/test/watch_list.shtm
Description: clickjacking essentially allows an attacker to disguise UI elements (usually an invisible iframe) as something else, thus tricking the user into clicking something [8]. In the example provided in [8], the decoy content would be a `div` with contents that say “click this to win something”, while the `iframe` is hiding a payment button that overlaps the “win” button.

OWASP has found that the Mango application has nothing that prevents this type of attack.

Solution: OWASP recommends setting Content-Security-Policy and X-Frame-Options HTTP headers for all pages returned by the backend [1]. SAMEORIGIN allows the page to be used in an iframe from the same server, while DENY makes it impossible to put the page in an iframe.

2.2.8 Cookie without SameSite Attribute

Risk level: low
Category from OWASP top 10: A01:2021 - Broken Access Control
URL:
http://localhost:8080/test/
Description: This applies to the cookie JSESSIONID, which is used by the Mango application. Without an appropriate SameSite attribute, this cookie can be used for malicious requests [1].

Solution: According to OWASP, we can set the SameSite attribute to “strict” or to “lax”[1]. If the value is “strict”, the cookie will not be transmitted if the origin of the site is not the same as the destination (for example, if the cookie is for GitHub, it will not be transmitted if the user follows a link to GitHub from another site; the user will need to login again); lax means that it can be transmitted if following a link from another website but will not be transmitted in a POST request [10].

2.3 Manual for Penetration Testing

After setting up Firefox to use ZAP as a proxy and starting Mango we took the following steps to perform the penetration testing:

a. Setting up authentication (based on this guide)
   - Open Firefox using the Firefox icon on upper toolbar
   - Log into Mango using username and password
   - Create new context on top note of Mango site
   - Find login POST request and “Flag as context” / ”Form-based Auth Login request”
   - Add regex pattern to detect if logged in
   - Add user in user tab with username=”admin” and password=”admin”
   - Turn on forced user by clicking on lock at top of application
b. Starting an automated scan
   - Right click on “test”-node in Mango-node-tree and select Attack / ActiveScan

- Have patience *(depending on the settings, this can take a few hours (is links with long polls are excluded) to 15+ hours (if links with long polls are included)*

c. Manual walkthrough
   - Manually navigating through the website, e.g. entering a valid SQL-query

2.4 Challenges and Solutions

Authentication: One challenge we faced was authentication. The web crawler used by ZAP goes through the website, checking for vulnerabilities. The problem is that many websites, like the Mango web interface, require the user to authenticate before they allow access to some parts of the website. That means that ZAP is not able to scan the biggest part of the website. Even though ZAP detects the authentication input fields and tries to enter random strings, these do not give access. To solve this authentication problem the pentesters, us, have to give ZAP valid authentication credentials. That way, ZAP can log in and crawl all parts of the website. The instructions to set up ZAP for authentication are given in section 2.3.

Manual Walkthrough: After authentication, the automated web crawler had access to all sections of the website a user has access to. For some parts of the website, it made sense to manually give some input. One example is the SQL-section. Even though ZAP can enter random strings into the SQL-query-text field, these do not result in valid outputs from the database, which means that the scan is still missing certain areas. The solution to this problem is to walk through some parts of the website manually while ZAP scans in the background. For the SQL-section, for example, we manually submitted valid SQL-queries while ZAP was scanning.

3 Comparison of Results

Both types of security vulnerability detections provide us with interesting results. While the static analysis shows us that the code is poorly written, the penetration tests show that many high and medium risks were present in the code. In terms of numbers, a total of 29 alerts were found with the penetration test approach, while the static analysis highlighted 3.6 thousand bugs in the code. In this aspect, static analysis can return many more problems with the code than a penetration test, but the reason behind this difference is that static analysis can return many false positives that need to be sifted through and verified manually. Furthermore, penetration testing is only front-impact testing, so it usually returns fewer vulnerabilities than static analysis.

In terms of CWE categories, both methods can provide similarities. For example, many risks found from the penetration test are classified as “A03 – Injection” and can also be found with static analysis (1.2.1 Cross-Site Scripting Vulnerability). However, some risks were only found with the penetration test (e.g. A01 – Broken Access Control and A04 - Insecure Design). These differences can be explained by the fact that static analysis cannot detect issues due to flaws in the design of Mango, while the penetration test can, as it tests the code that is actually being exposed.

4 Conclusion

In this report, we investigated the security aspect of the entirety of the Mango system using static analysis and a penetration test. We showed 8 separate vulnerabilities detected by the static analysis and 8 more vulnerabilities detected by the penetration test. The tools used for this report were the SonarCloud software for the static analysis and OWASP Zap to run the penetration test. The first result obtained was that Mango’s code could be very poorly written, if most of the bugs detected by SonarCloud aren’t false positives. This will require further work to manually validate these results. The other results, obtained with the penetration, were that 29 risks are present, with varying levels of severity. A comparison of both methods showed that neither method is objectively better than the other. Of course, while many risks and bugs can be detected with these methods, they are not a substitute for actual source code review, but this requires highly skilled security aware developers.

5 References

[1] OWASP Foundation. (2022) OWASP Zap (Version 2.12.0). [Online]. Available: https://owasp.org/www-project-zap/

[2] J. Charles, “XSS aka HTML injection attack explained,” Medium, 28-Mar-2019. [Online]. Available: https://medium.com/@jamischarles/xss-aka-html-injection-attack-explained-538f46475f6c . [Accessed: 28-Nov-2022]

[3] Imperva “Command injection,” *Command Injection | Imperva*. [Online]. Available: https://www.imperva.com/learn/application-security/command-injection/. [Accessed: 28-Nov-2022].

[4] Invicti Security, “Types of SQL Injection (SQLi),” *Acunetix*, 15-Jul-2020. [Online]. Available: https://www.acunetix.com/websitesecurity/sql-injection2/#:~:text=What%20is%20a%20time%2Dbased,query%20is%20true%20or%20false. [Accessed: 28-Nov-2022].

[5] SQLINJECTION.NET, “Time-based blind SQL injection attacks,” *SQL Injection*. [Online]. Available: https://www.sqlinjection.net/time-based/. [Accessed: 28-Nov-2022]

[6] MDN contributors, “Apache Configuration: .htaccess,” *MDN Web Docs*, 18-Nov-2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Apache_Configuration_htaccess. [Accessed: 28-Nov-2022].

[7] A. Politis and R. Osipov, response to “How to prevent a file from direct URL Access?”, *Stack Overflow*, 10-Feb-2017. [Online]. Available: https://stackoverflow.com/a/10236791/11627201. [Accessed: 28-Nov-2022].

[8] PortSwigger , “Clickjacking (UI redressing),” *PortSwigger Academy*. [Online]. Available: https://portswigger.net/web-security/clickjacking. [Accessed: 28-Nov-2022].

[9] ISO/IEC. 25010 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. 2011

[10] P. Krawczyk, Riramar, N. Smithline, D. Wetter, and Ayesh, “SameSite,” *OWASP Foundation*. [Online]. Available: https://owasp.org/www-community/SameSite. [Accessed: 29-Nov-2022].