Software Quality Engineering Heng Li

Security Mango

Group One

Felipe Manoel - 2230852

Hsiu-Tsu Shui – 2229737

Kun-Che Tsai – 2229658

Koen van Oijen – 2229603

Diederik Ponfoort - 2230929

Contents

Abstract **3**

Introduction **3**

Static Analysis **3**

Summary of the analysis 4

8 vulnerabilities 4

Security Hotspots 6

Process description 7

Encountered difficulties 8

Penetration Testing **9**

A. Manual 9

B. Alerts 10

Comparison Between Static Analysis and Penetration Testing **14**

Conclusion **15**

Abstract

This report is regarding the security objectives and security assurance process of the Mango system. Security should be one of the main concerns for a M2M system as they have access to a lot of (sensitive) data and sometimes can alter the behavior of real-time devices. This paper presents two types of analyses: static analysis and penetration testing.The static analysis found quite some bugs/vulnerabilities and a few security hotspots. The penetration testing found 19 alerts for the security of Mango. The paper describes several of these problems and further suggests possible solutions. Finally, the paper described the main differences of the two different methods.

Introduction

In the last assignment for this course, the security of Mango will be analyzed. The open-source Mango software is one of the crucial elements of a whole M2M system. Its function is to gather and process the data, then present the raw data in a comprehensible way to users. In the previous assignments, we already reviewed some quality characteristics and the performance efficiency of Mango. From these previous analyses, it was already clear that the Mango system was not perfect yet and some suggestions were given. Software security focuses on vulnerabilities and prevents third parties from making advantage of these mistakes. It is an important analysis to perform especially for a system like Mango. Since Mango has access to a lot of data from sensors and is able to automatically perform actions with the real word. To detect the vulnerabilities we will perform two types of analysis on Mango. We will start with a static analysis which will be performed with the aid of SonarCloud. A static analysis is an automatic test that runs through the code and checks for vulnerabilities based on known rules and templates. This is a fast method of finding vulnerabilities, but cannot find flaws bounded in design and shows a lot of noise.

The second test described in this paper is a penetration test. This method can also be called ethical hacking and the tester tries to find the weakness themselves. With this method you only will evaluate the code that is visible for the users. However, this means the user will not find vulnerabilities in the backend. For penetration testing the OWASP Zap tool will be used. Finally, at the end of the paper, a comparison of the two methods will be presented.

Static Analysis

In this chapter, the results of the static analysis with Sonarcloud are presented. The section is divided into three parts. First, a small overview is presented of the overall results. Then 8 specific examples are highlighted, and last, our own experience of the Sonarcloud process will be shared.

Summary of the analysis

As visible from the static analysis, there are quite a lot of issues and vulnerabilities inside the Mango system. The reliability rating based on the static analysis is E, with 3.6k bugs found in the Mango system.The maintainability of the system is rated as an A. Moreover, 4.4k code smells are found within the Mango system. The security rating of the system is E, with 3 major security vulnerabilities found that can be exploited by hackers. Additionally, 47 security hotspots are found inside the Mango system and 25% of the system is duplicated code. To conclude, the Mango system has quite a lot of vulnerabilities, security hotspots and is overall not well protected.

8 vulnerabilities

1) **Code reflects user controlled data**
    - **Description + risks:** User-provided data should typically be seen as untrusted. In this example, the user-provided data gets used by the code which could cause issues. This leaves the system vulnerable to HTTP requests with malicious content, and makes it possible to be used as a sink, which would allow malicious values as arguments. This leaves the system vulnerable to XSS-attacks.
    - **File of vulnerability:** build/resources/dojo/tests/widget/showPost.php
    - **Severity level:** blocker
    - **Type of the vulnerability according to CWE:** MITRE, CWE-79 - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
    - **Recommendation for a solution:** Validate user-provided data, this could be done by creating a whitelist of the data inputs that are allowed and blocking all other inputs.
2) **Commented out code**
    - **Description + risks:** Sections of code are commented out. This bloats programs and reduces readability. Moreover, unused code should be deleted and retrieved from source history if needed later. There are no big risks involved, it would just improve readability if the comments were removed.
    - **File of vulnerability:** build.xml
    - **Severity level:** major
    - **Type of vulnerability according to OWASP, SANS, or CWE:** CWE-1041 - Use of redundant code
    - **Recommendation for a solution:** Delete the commented out code from the source files
3) **switch statement should have default clauses**
    - **Description + risks:** The switch statement in this file does not have a default clause. A default clause is a form of defensive programming. Even if the switch covers all current values (enum), a default clause is necessary, because there is no guarantee that the enum does not get extended. In the worst case this could cause system failure.
    - **File of vulnerability:** build/resources/dojo/tests/rpc/JSON.php
    - **Severity level:** critical

- **Type of vulnerability according to OWASP, SANS, or CWE:** MITRE, CWE-478 - Missing Default Case in Switch Statement
- **Recommendation for a solution:** Add a default clause to the switch statement.

4) **Double checked locking**
   - **Description + risks:** Double checked locking is the bad practice of checking an initialized object state before and after a synchronized block is entered to decide to initialize the object. This makes it possible for a second thread to use the uninitialized object, which in the worst case could cause system or program failure.
   - **File of vulnerability:** src/com/serotonin/mango/Common.java
   - **Severity level:** blocker
   - **Type of vulnerability according to OWASP, SANS, or CWE:** MITRE, CWE-609 - Double-checked locking
   - **Recommendation for a solution:** There are multiple ways to fix this. It is easiest to not use double checked locking and synchronize the method as a whole instead.
5) **Strings and boxed types comparing**
   - **Description + risks:** Comparing two instances of a String type or boxes integer type using == or != is a common bad practice. This is because it is not comparing the actual value, but the location of the memory
   - **File of vulnerability:** ..com/serotonin/mango/vo/event/MaintenanceEventVO.java
   - **Severity level:** Major
   - **Type of vulnerability according to OWASP, SANS, or CWE:** MITRE, CWE-595 - Comparison of Object References Instead of Object Contents
   - **Recommendation for a solution:** You can use the function .equals(). This will actually check the value instead of the location in memory.
6) **"InterruptedException" should not be ignored**
   - **Description + risks:** In the file a interruptedException is being caught. However, once logged nothing will be done with it. Simply using the InterruptedException will clear the interrupted state and information can get lost if not handled properly.
   - **File of vulnerability:** src/com/serotonin/mango/rt/dataSource/galil/GalilDataSourceRT.java
   - **Severity level:** Major
   - **Type of vulnerability according to OWASP, SANS, or CWE:** MITRE, CWE-391 - Unchecked Error Condition
   - **Recommendation for a solution:** Either rethrow the InterruptedExceptions or kill the thread completely.
7) **Unused assignments should be removed**
   - **Description + risks:** In the ReportsDwr file, a new variable user get created without being used in the nearby function. This is a waste of storage and could lead to errors.
   - **File of vulnerability:** src/com/serotonin/mango/web/dwr/ReportsDwr.java
   - **Severity level:** Major
   - **Type of vulnerability according to OWASP, SANS, or CWE:** MITRE, CWE-563 - Assignment to Variable without Use ('Unused Variable')

- **Recommendation for a solution:** To prevent any waste of resources, the best way is either to use the new created variable or remove it all together.

8) **Standard outputs should not be used to directly log anything**
   - **Description + risks:** In this file the developers want to print to StartsAndRunsTimeList and see the results with System.out. However, this makes it harder to retrieve the logs, read the log and this way it won't be recorded.
   - **File of vulnerability:** src/com/serotonin/mango/view/stats/StartsAndRuntimeList.java
   - **Severity level:** Major
   - **Type of vulnerability according to OWASP, SANS, or CWE:** OWASP Top 10 2021 Category A9 - Security Logging and Monitoring Failures
   - **Recommendation for a solution:** They should use a logger. This way they can recall the loggings and access it easily.

Security Hotspots

1) **Dynamically formatted SQL query**
   - **Description+Risks:** The developers added an SQL query that is created dynamically formatted. Dynamically formatted SQL queries are hard to maintain and to debug, which could result in a SQL Injection by adding a value which was received by the function to the query.
   - **File of Vulnerability:** src/com/serotonin/mango/db/DBConvert.java
   - **Review Priority:** High
   - **Type of Possible Vulnerability:**OWASP Top 10 2021 Category A3 - Injection
   - **Recommendation for a solution:** They could use parameterized queries with prepared statements instead.
2) **Pseudorandom number generator**
   - **Description+Risks:** A pseudorandom number generator is used in the application. This type of generator can be predicted by an attacker, who can use this number to affect other users or to affect sensitive data.
   - **File of vulnerability:** src/com/serotonin/mango/rt/dataSource/virtual/ChangeTypeRT.java
   - **Review Priority:** Medium
   - **Type of Possible Vulnerability:** OWASP Top 10 2021 Category A2 - Cryptographic Failures
   - **Recommendation for a solution:** They could use a cryptographically strong random number generator instead of pseudorandom number generator. They could use the class java.security.SecureRandom for example.
3) **Debug Configuration Enabled**
   - **Description+Risks:** The developers left some prints which are used to debug in the middle of the production code. Debug configuration is secure sensitive as it may give detailed information about how the application runs that may facilitate an attack.
   - **File of Vulnerability:** src/br/org/scadabr/rt/dataSource/dnp3/Dnp3DataSource.java
   - **Review Priority:** Low

- **Type of Possible Vulnerability:** OWASP Top 10 2021 Category A5 - Security Misconfiguration
- **Recommendation for the solution:** They could use loggers instead of printStackTrace.

4) **Non-Null Window Opener Attribute**
   - **Description+Risks:** The developers opened a new window in a html file without setting the window.opener attribute to null. As a consequence, an attacker can use the data from window.opener to change the original page information and cause problems to the user as they may think that it is the same page they accessed before.
   - **File of Vulnerability:** build/WEB-INF/snippet/eventList.jsp
   - **Review Priority:** Low
   - **Type of Possible Vulnerability:** OWASP Top 10 2021 Category A5 - Security Misconfiguration
   - **Recommendation for the solution:** They can use rel=”noopener” to force the window.opener to be null in the opened page.
5) **Resource Integrity Feature**
   - **Description+Risks:** The HTML file gets a JavaScript script from a webpage. AS a result, if the resource is changed by a malicious one, the mango application will be compromised too.
   - **File of Vulnerability:** build/resources/dojo/tests/data/old/test_data.html
   - **Review Priority:** Low
   - **Type of Possible Vulnerability:** OWASP Top 10 2021 Category A8 - **Software and Data Integrity Failures**
   - **Recommendation for the solution:** The developers can implement a resource integrity check for all static resources.

Process description

The results described above were founded with SonarCloud, an online service for detecting security vulnerabilities and catching bugs. It is mainly used in combination with Github or other pull-based development programs. Getting the results with SonarCloud was no easy ride and took some effort. However, once figured out (with the help of) the results can be easily replicated. In this section, we give a short step-by-step description of the final process and finally we will discuss some of the issues we encountered on the way.

Guide

1. Fork the repository you want to check (in our case Mango)
2. Login on Sonarcloud with your GitHub account
3. Find the plus button, select “Analyze new project” and choose the project you want to analyze
4. You will be brought to a configuration page. There you select Manually
5. Pick Others (for JS, TS, GO, Python, PHP, ..)

6. Pick your OS (windows in our case)

7. Download and unzip SonarScanner for Windows, add the bin directory to the Path in the environment variable

8. Add SONAR_TOKEN to the environment variable as well with the value a code given by Sonarcloud

9. Lastly run the following command in the projects folder:

```
sonar-scanner.bat \ -D"sonar.organization=koenvanoijen" \
-D"sonar.projectKey=koenvanoijen_mango" \ -D"sonar.sources=." \
-D"sonar.host.url=https://sonarcloud.io" \ -Dsonar.java.binaries=src/com/
```

10. The results will appear on SonarCloud

Encountered difficulties

Like mentioned before, the process of getting to the results was not without problems. There were 3 major problems we ran into. First, the installation went very smooth, but we encountered some issues with Java. Mango needed java 8 and that was the version we had available on our computer. However, Sonarcloud required at least java 11. This problem was easily fixed when recognized by updating our Java. The second problem was caused by the fact that we followed the documentation available on Moodle. This pushed us towards using Maven, which was needed last year. Since Mango is built with Ant this did not work as planned. Sonarcloud did work, but only showed us the results of the created POM file. When we realized that Maven was not going to work, we decided to go for the ‘other’ build option and followed the guidance of Sonarcloud. However, this returned an error mentioning that it could not handle Java files. Finally, we added the line “-Dsonar.java.binaries=src/com/” and this fixed the error and gave us the results described above.

Penetration Testing

In this chapter, our goal is to identify the vulnerabilities through penetration testing, which simulates malicious attacks against the Mango system.

A. Manual

In our previous two TPs, the Mango system could be launched by running the startup.bash and creating the localhost on our own computer. To execute the penetration testing, we have to establish a via point to monitor the system's behavior whenever a request is sent to the system. That’s how the ZAP works basically. Therefore, upon launching the ZAP, it asks for a port to operate on. As 8080 is reserved for the Mango, we chose 8081 as our port for penetration testing.

After the installation and the setup were done, we refreshed the ZAP. Once we are on the ZAP main page, open the firefox browser, which is a quick start to access the Mango system, and start the proxy to run the penetration testing and observe the alert. The step to execute the penetration test is selecting the traditional spider to crawl the website, and the alerts will be automatically generated at the Alerts tab. Nonetheless, the scan can only detect the watch list as default. If we want to know other potential alerts existing in other functionalities, we have to run the manual scan by pasting the URL of the feature to the starting point in the Spider tab. Hence, the results can be more complete.

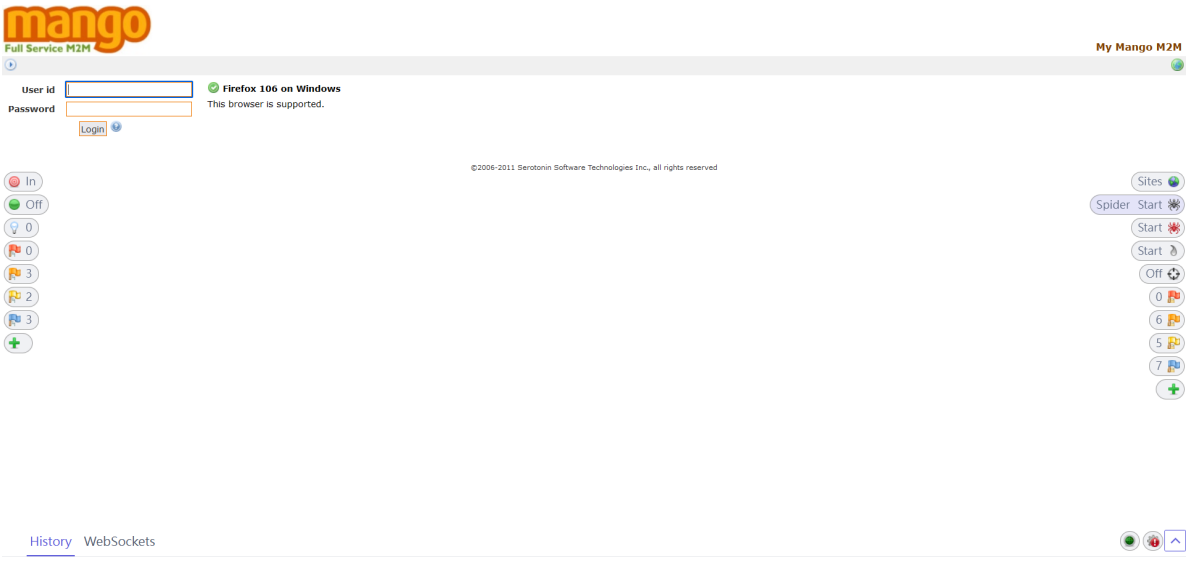

figure1 Logging Page of Mango after Connecting to the Proxy

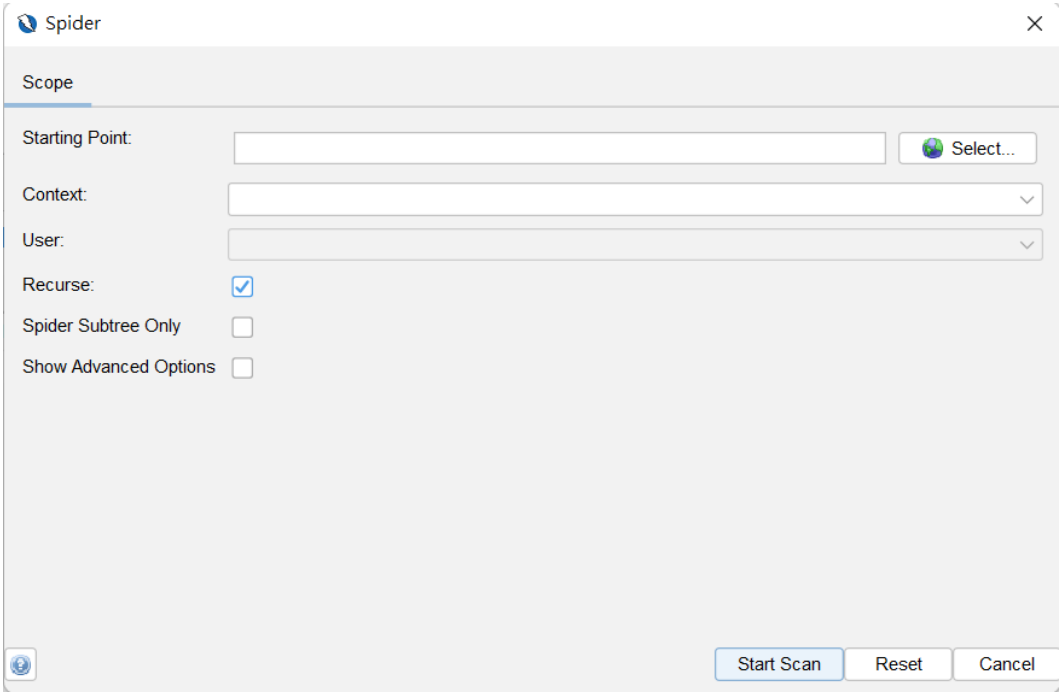

figure2 Manual Scan Snippet

B. Alerts

Throughout the penetration testing, 19 alerts appeared. So, we picked 8 out of 19 to inspect the details of the potential vulnerabilities.

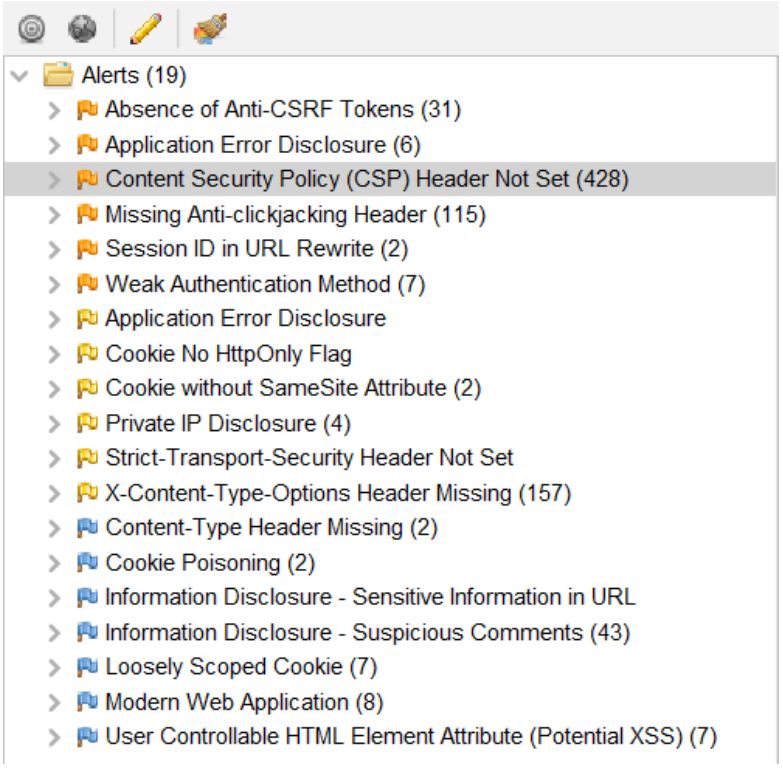

figure3 Alerts

a. Absence of Anti-CSRF Tokens
   - Risk: Medium

- OWASP: **A01 - Broken Access Control**
- CWE: **CWE352 - Cross-Site Request Forgery (CSRF)**
- URL: https://cwe.mitre.org/data/definitions/352.html
- Description:
  No Anti-CSRF tokens were found in a HTML submission form.
  A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a website has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has in a website. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.

  CSRF attacks are effective in a number of situations, including:
  * The victim has an active session on the target site.
  * The victim is authenticated via HTTP auth on the target site.
  * The victim is on the same local network as the target site.

  CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.

- Solution:
  Phase: Architecture and Design
  Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
  For example, use anti-CSRF packages such as the OWASP CSRFGuard.

  Phase: Implementation
  Ensure that your application is free of cross-site scripting issues because most CSRF defenses can be bypassed using an attacker-controlled script.

  Phase: Architecture and Design
  Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330). Note that this can be bypassed using XSS. Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.
  Note that this can be bypassed using XSS. Use the ESAPI Session Management control. This control includes a component for CSRF. Do not use the GET method for any request that triggers a state change.

  Phase: Implementation

Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referrer for privacy reasons.

b. Session ID in URL Rewrite
   - Risk: Medium
   - OWASP: **A01 - Broken Access Control**
   - CWE: **CWE200 - Exposure of Sensitive Information to an Unauthorized Actor**
   - URL: https://cwe.mitre.org/data/definitions/200.html
   - Description:
     URL rewrite is used to track user session ID. The session ID may be disclosed via a cross-site referer header. In addition, the session ID might be stored in browser history or server logs.
   - How to attack Mango
     A malicious user can steal other people's session from the URL and hijack the session. Once the attacker knows the session ID, he or she can modify the session id stored in the cookie and bypass the authentication without the username and password.
   - Solution:
     For secure content, put the session ID in a cookie. To be even more secure consider using a combination of cookie and URL rewrite.

c. Application Error Disclosure
   - Risk: Medium
   - OWASP: **A01 - Broken Access Control**
   - CWE: **CWE200 - Exposure of Sensitive Information to an Unauthorized Actor**
   - URL: https://cwe.mitre.org/data/definitions/200.html
   - Description:
     This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.
   - How to attack Mango
     The error page discloses that the Mango is implemented in Java. Whatsmore, it also specifies the Java version and the library being used. A malicious attacker can launch attacks with known vulnerabilities based on the sensitive information. Worse still, an error page may show the key or seed that are used to encrypt the message, which provides hackers a opportunity to decode the message or even launch a man in the middle attack.
   - Solution:
     Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.

d. Weak Authentication Method
   - Risk: Medium
   - OWASP: **A02 - Cryptographic Failures**

- CWE: **CWE326 - Inadequate Encryption Strength**
- URL: https://cwe.mitre.org/data/definitions/326.html
- Description:
  HTTP basic or digest authentication has been used over an unsecured connection. The credentials can be read and then reused by someone with access to the network.
- Solution:
  Protect the connection using HTTPS or use a stronger authentication mechanism.

e. Missing Anti-clickjacking Header
- Risk: Medium
- OWASP: **A04 - Insecure Design**
- CWE: **CWE1021 - Improper Restriction of Rendered UI Layers or Frames**
- URL: https://cwe.mitre.org/data/definitions/1021.html
- Description:
  The response does not include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options to protect against 'ClickJacking' attacks.
- Solution:
  Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app. If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise, if you never expect the page to be framed, you should use DENY. Alternatively, consider implementing Content Security Policy's "frame-ancestors" directive.

f. Content Security Policy (CSP) Header Not Set
- Risk: Medium
- CWE: **CWE693 - Protection Mechanism Failure**
- URL: https://cwe.mitre.org/data/definitions/693.html
- Description:
  Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images, and embeddable objects such as Java applets, ActiveX, audio and video files.
- Solution:
  Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header, to achieve optimal browser support: "Content-Security-Policy" for Chrome 25+, Firefox 23+ and Safari 7+, "X-Content-Security-Policy" for Firefox 4.0+ and Internet Explorer 10+, and "X-WebKit-CSP" for Chrome 14+ and Safari 6+.

g. Cookie No HttpOnly Flag
- Risk: Low
- OWASP: **A05 - Security Misconfiguration**

- CWE: **CWE1004 - Sensitive Cookie Without 'HttpOnly' Flag**
- URL: https://cwe.mitre.org/data/definitions/1004.html
- Description:
  A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.
- How to attack Mango
  An attacker can embed javascript code in the input text box and test if the script can be executed. If the script can be executed, the attacker can launch a further attack by stealing one's sensitive information such as the admin's cookie session by XSS attack. Once successful, the attacker can hijack the admin's session and break the access control.
- Solution:
  Ensure that the HttpOnly flag is set for all cookies.

h. Cookie Poisoning
- Risk: Informational
- OWASP: **A03 - Injection**
- CWE: **CWE20 - Improper Input Validation**
- URL: https://cwe.mitre.org/data/definitions/1004.html
- Description:
  This check looks at user-supplied input in query string parameters and POST data to identify where cookie parameters might be controlled. This is called a cookie poisoning attack and becomes exploitable when an attacker can manipulate the cookie in various ways. In some cases this will not be exploitable, however, allowing URL parameters to set cookie values is generally considered a bug.
- Solution:
  Do not allow user input to control cookie names and values. If some query string parameters must be set in cookie values, be sure to filter out semicolon s that can serve as name/value pair delimiters.

Comparison Between Static Analysis and Penetration Testing

As can be seen in the former sections, we got different results from the two methodologies adopted in this report. In the static analysis, we got a high number of issues, including 3600 bugs, 3 vulnerabilities, 4400 code smells and 47 security hotspots. In the penetration tests, we got a much smaller number of results, with 19 alerts.

In relation to the types of vulnerabilities, they do not have any intersection in terms of CWE categories, however it was common for penetration testing to have OWASP A01 - Broken Access Control.

The main reason for these differences is the way which one of these strategies found the vulnerabilities. The static analysis tries to find common patterns inside the code of the application that may reveal a recurrent mistake that might affect the application. This

analysis is made over the entire system, so it explains it results in a huge number of issues, as mentioned before. However, this approach has a weak side; namely the existence of false positives. A reported code snippet might not cause a serious problem. In the penetration testing, the application is tested by realizing specific attacks, in our case, the ZAP application, which was used in the penetration testing, stay between the server and the user while handling the requests, this might explain why this strategy caught more OWASP A1 issues as it they are mostly authentication issues which are closer to the ZAP application. This approach tries very specific attacks to application, which explains the low number of alerts found in comparison to the static analysis. This follows the remark Gary McGraw makes concerning penetration testing's weak side: ".... can only identify a small representative sample of all possible security risks in a system"..

Conclusion

In this paper, firstly we performed a static analysis and found a huge amount of issues including code smells bugs and vulnerabilities. We also suggested how to solve problem to Mango application. We also did a penetration testing and found more vulnerabilities by connecting the mango application to ZAP, as did before we discussed them and showed a proper solution.

Finally, we compared the results, confirming our expectation that the static analysis would have a wider range of results, with the drawback of having false positives and the penetration testing finding specifics vulnerabilities, but with the disadvantage of recognizing a smaller number of vulnerabilities.