POLYTECHNIQUE MONTRÉAL

LOG8371E: Software Quality Engineering

TP3: Security

Submitted by:

Abderrahim Ammour	1924705
Olivier Lecavalier-Hurtubise	1957948
Kyrollos Bekhet	1949242
Khadija Rekik	1921994
Reetesh Dooleea	1957959

December 1st, 2022

Table of contents

1. Abstract

The objective of this report is to get familiar with the software security objectives and security assurance process, identification of security vulnerabilities through static analysis of the source code, and identification of security vulnerabilities through penetration testing. First, this report is divided into three main sections, the static analysis in which a summary of the results is described, 8 comments on found vulnerabilities are provided as well as a manual to reproduce the static analysis of the source code of the Mango System. Then the penetration section contains a manual to reproduce the penetration tests, the test result as well as some recommendations to solve the security problems in the application. Finally, the last section includes a comparison between static analysis and penetration testing and also a discussion about the difference between those two approaches. SonarCloud technology will be used for static analysis and the OWASP ZAP tool will be used for penetration testing.

2. Introduction

The motivation of this work is to understand software quality assurance practices and to design software quality assurance plans. Another purpose is to develop verification strategies. SonarCloud is used to evaluate the security aspect of the entire source code of the Mango system automatically. In fact, sonarCloud can be configured to run automatically when there is a change in the source control system.

Firstly, we will perform a static analysis with the help of the sonarCloud software. Static analysis allows us to examine the code without having to actually run the program. It is based on specified rules and templates. As a matter of fact, static analysis allows us to detect software quality issues such as code errors as well as security issues and possible hotspots which can cause vulnerabilities in the system. It has many advantages, indeed, it effectively allows us to rapidly check that the code is of high quality and that it achieves regulatory compliance. In fact, developers can comprehensively test their code in a non-runtime environment and check that their code standards are met within sonarCloud. As a consequence, this allows higher-quality code to reach testers faster and furthermore it leads to an accelerated software development life cycle. At the same time, the static analysis still has notable disadvantages: for example, it tends to produce many false positives which do require a significant amount of manual effort to validate and fix.

Secondly, penetration testing is executed on the schedule events feature of the deployed Mango system using the ZAP tool. Penetration testing allows us to simulate a malicious attack in order to identify the business impacts of the system's vulnerabilities. It is a form of black-box testing using the same tools, techniques and processes as regular attackers.

Finally, our end goal is to compare the results from the static analysis and penetration testing in order to understand the difference and scope of the two methods.

3. Static analysis

3.1. Summary of the results,

To begin with, SonarCloud is used to do the security static analysis for the Mango source code. After doing the security analysis on the source code of the whole project, sonarCloud gave us a summary of the results [5]. In this summary, indicators for different quality aspects like Reliability, Maintainability, Security, Security Review and Duplications were found. For reliability, sonarCloud found 2100 possible bugs. For maintainability, there are 18000 code smells and 89.6% code duplication. As for security, SonarCloud found 2 vulnerabilities and 297 security Hotspots for security review. Table 3.1 presents the summary provided by sonarCloud after the analysis.

Tableau 3.1: Summary table extracted with sonarCloud

Quality aspect	Value
Reliability	2100 bugs
Maintainability	18000 code smells
Security	2 vulnerabilities
Security Review	279
Duplications	89.6%

As noticed, MangoSource code has two vulnerabilities and a lot of security hotspots. In the next subsection, the two blocker vulnerabilities as well as six other security Hotspots of low or medium priority are detailed by analyzing each vulnerability and providing comments on them as well as recommendations to fix them. In the analysis, the location of the vulnerability is identified as well as its type, its severity level, and its root cause.

3.2. Comments for the vulnerabilities

In this section, eight vulnerabilities are identified. For each vulnerability, a description will be provided, and the file of the vulnerability, its severity level, its potential risk as well as its type according to OWASP, SANS or CWE will also be mentioned. Finally, each vulnerability will be accompanied by a recommendation as to how to solve or eliminate the problem. Moreover, SonarCloud separates the severity criteria into three categories: blocker, critical and major. The vulnerabilities can also be found under the Security Hotspots section where the categories are low, medium and high.

1. Dynamic code execution is vulnerable to injection attacks

This vulnerability is caused by the execution of dynamic code without verifying the user inputs. This vulnerability may lead to a loss of sensitive data since a user may enter malicious code that will be executed dynamically. This vulnerability has a severity level of Blocker and it belongs to the types: OWASP-a1, CWE-94 and SANS-TOP25-risky. For instance, the vulnerability can be found in *mangoSource/war/resources/dojo/src/io/RepubsubIO.js* file. A solution to this vulnerability is to do a whitelist of allowed values, in this way the malicious code is not executed.

2. Endpoints are vulnerable to cross-site scripting attacks (XSS)

This vulnerability is caused by trusting User-provided data that should not be trusted such as URL parameters, POST data payloads or cookies. This vulnerability happens because of the way HTTP requests are processed. 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 in a behaviour called Reflection. Endpoints reflecting untrusted data could allow attackers to inject code that will be executed in the user's browser. This could lead to serious attacks like accessing/modifying sensitive information or impersonating other users. This vulnerability belongs to the Blocker severity level and it belongs to the types: OWSAP-a7, SANS-TOP25-insecure and CWE. For instance, the vulnerability can be found in *showPost.php* under *mangoSource/war/resources/dojo/tests/widget/* folder. A solution to this vulnerability is to validate user-provided data based on a whitelist and reject input that is not allowed.

3. Insecure implementation of HTML anchor tag causing phishing attacks

This vulnerability is part of the low priority level under the Security Hotspots section. The risk is located at line 120 in *mangoSource/resources/WEB-INF/snippet/eventList.jsp* file. Moreover, the type of the vulnerability is *CWE-1022* and OWASP-A6. It is when an anchor tag of an HTML element doesn't contain the *noopener* attribute. For instance, the current risk in the current file is: <a href="http://mango.serotoninsoftware.com/download.jsp" target="_blank">...</a>. We can notice that the HTML element contains only the HREF attribute. This consists of a risk as the linked website, indicated via the HREF attribute, could hijack or overtake the control of the linking page tab in a Web browser and redirect the user to a phishing page. By doing so, the user might enter confidential or personal information such as credentials which could be stolen by the attackers. The malicious website could also prompt the user to perform malicious actions such as downloading and installing malware and so on. Furthermore, this scenario occurs when a user opens a link from the base website. The newly opened window has access to the original window. A hacker could override the link back to the original or initial page in order to cause a redirection to a fake page. Hence, this constitutes a big vulnerability for an end user. To solve this problem, whenever an anchor tag is used or created, it is essential to add the attribute *noopener* to the HTML element as follows: *rel="noopener"*. This eliminates the possibility of phishing attacks from the target page to the original page.

4. Usage of HTTP protocol

This vulnerability from the Security Hotspot section is related to the use of hard-coded IP addresses in the source code. This risk can be found in the *dojo.js* file in *mangoSource/war/resources/dojo* folder. The severity level of this risk is low and the type of vulnerability is OWASP-A3. This is a vulnerability since HTTP is used to communicate with an external API in the code. Hence, this scenario is security-sensitive because the HTTP protocol doesn`t have an implemented mechanism to encrypt the transferred data so the data is transferred in clear text. This can lead to the exposure of sensitive data if an attacker intercepts and listens over the network. To eliminate this risk, data should transit over a secure and authenticated protocol like TLS which is used in HTTPS.

5. Dynamic code execution is vulnerable to injection attacks

This vulnerability from the Security Hotspot section is related to the dynamic execution of user input data by creating a function without doing any verification. This vulnerability can lead to the execution of malicious code which can be used to access or modify sensitive information. This vulnerability belongs to the code injection section and it corresponds to CWE-94 and OWASP-a1 vulnerability types and has a medium priority level. In the past, this issue has led to those vulnerabilities: *CVE-2017-9807* and *CVE-2017-9802*. The vulnerability can be found in *mangoSource/war/resources/dojo/dojo.js* file line 9338. To solve or mitigate this problem we can run this code in a sandboxed environment by using iframes and SOP. In this way, the malicious code doesn't affect the rest of the system.

6. Weak Cryptography

This vulnerability from the Security Hotspot security is related to the use of a number generated by a pseudorandom number generator. *Math.random()* function used is a weak pseudorandom number generator which makes the next value predictable. An attacker may guess the next value and impersonate another user to access sensitive information. This issue has led to these vulnerabilities in the past: CVE-2013-6386, CVE-2006-3419 and CVE-2008-4102. This vulnerability is from the Weak Cryptography category and of type CWE-338 and it has a medium priority level. The vulnerability can be found in *mangoSource/war/resources/dojo/src/collections/SkipList.js* file line 76. To solve or mitigate this vulnerability, we can use a strong pseudorandom number generator like *crypto.getRandomvalues()*.

7. Inefficient regular expression complexity

Another vulnerability from the Security Hotspot section is of type medium priority and belongs to the Denial of Service (DoS) category. It lies in the *docs.js* located in the *mangoSource/war/resources/dojo/src/* directory at line 36. The severity level is medium and the type of vulnerability corresponds to CWE-1333. In our case, the problem with the regular expression, which is /^\s+|\s+$/g, is risky as this causes performance issues in our application. For this process to work, regular expressions usually use backtracking to evaluate an input or a string. Hence, in some cases, the complexity of the regular expression is so high that it can lead to catastrophic backtracking. Thus, this can cause a denial of service for the application. This vulnerability is present whenever a user has access to an input or when a regular expression is applied but there is no timeout in place to limit its evaluation time. To solve the problem, we should first have a better understanding of how regular expression works. Then, we should make sure that the regular expression isn't followed by any pattern that can fail. This prevents backtracking from happening. An alternative way would be to try to validate the text without the use of regular expressions.

8. Sensitive data exposure

This last vulnerability from the Security Hotspots section concerns the usage of HTTP protocol rather than HTTPS. The name of the file where this vulnerability has been identified is *soundmanager2.js* which is situated in the *mangoSource/war/resources/* directory. The type of vulnerability is OWASP-A3 and CWE-319. Plus, its severity level is low. In this case, the vulnerability lies in the fact that requests are being made to an HTTP URL which is not totally safe. This vulnerability can be located at line 156 of the file: *var flashCPLink = 'http://www.macromedia.com/…*'. This consists of a risk as HTTP protocol lacks encryption of transported data. Moreover, it can't build an authenticated connection which allows hackers or middlemen to sniff the content being transported over the network. Hackers can intercept the network and cause all kinds of attacks. To prevent this from happening, it is essential to transmit data over HTTPS protocol since this is a secure, encrypted and authenticated protocol which allows data to be protected with a secured shell.

Table 3.2.1: Summary of the 8 highlighted vulnerabilities in the static analysis

Vulnerability number	**Vulnerability name**	**Type**	**File of the vulnerability**	**Potential risk**	**Severity/ Priority level**
#1	Dynamic code execution is vulnerable to injection attacks	OWASP-a1, CWE-94, SANS-TOP25-risky	mangoSource/war/resources/dojo/src/io/RepubsubIO.js	Unauthorized access or modification of sensitive information	Blocker
#2	Endpoints are vulnerable to cross-site scripting attacks (XSS)	OWSAP-a7, SANS-TOP25-insecure and CWE	mangoSource/war/resources/dojo/tests/widget/showPost.php	Accessing/modifying sensitive information or impersonating other users	Blocker
#3	Incorrect implementation of HTML anchor tag element	CWE-1022, OWASP-a6	mangoSource/resources/WEB-INF/snippet/eventList.jsp	Phishing attacks caused by external URLs	Low

#4	Usage of hard code IP addresses	OWASP-A3	mangoSource/war/resources/dojo/tests/validate/test_validate.js	Hard maintenance of the system and possibility of introducing bugs by not replacing all variables in a source base	Low
#5	Dynamic code execution is vulnerable to injection attacks	OWASP-a1, CWE-94, SANS-TOP25-risky	mangoSource/war/resources/dojo/dojo.js	Access or modification of sensitive informations	Medium
#6	Weak cryptography	CWE-338	mangoSource/war/resources/dojo/src/collections/SkipList.js	Impersonation attack to access sensitive information	Medium
#7	Inefficient regular expression complexity	CWE-1333	mangoSource/war/resources/dojo/src/docs.js	Denial of service attack	Medium
#8	Sensitive data exposure	OSWAP-A3, CWE-319	mangoSource/war/resources/soundmanager2.js	Sniffing content transmitted over the network	Low

3.3. Analysis performance manual

As mentioned before, SonarCloud is used to do the security static analysis. In order to set up SonarCloud we have followed [1] since we are most familiar with the GitHub interface. After logging in using a GitHub account we followed these steps to set up sonarCloud:

1. Choose to Analyse your first projects > Import an organization from GitHub.
2. Create the organization and Choose the project repository to analyze from Github. To use the free plan of sonarCloud, the repository had to be public.
3. After that we followed the setup instructions to create a secret token for sonarCloud on the GitHub account.
4. Following, the instruction steps a YAML file was created under the $REPO/.github/workflows so sonarCloud can run whenever a new code is pushed on the main branch.
5. A sonar-projet.properties file was created on the root of the project repo with the organization's name created previously.
6. Finally, the MangoSource code was pushed to a public repository as well as the newly created files to the repository so sonarCloud can execute the analysis.

4. Penetration testing

This type of testing is a process of simulating authorized attacks to discover security vulnerabilities in an application. Often referred to as pen testing or ethical hacking, it is usually performed according to the Open Web Application Security Project (OWASP) guidelines. In this section, penetration testing will be performed on the deployed Mango system using the open-source tool called ZAP [2]. The purpose of doing this type of testing is to identify exploitable vulnerabilities in networks, web applications, physical facilities and human resources to understand vulnerability to security threats and cyber-attacks better. In our situation, the focus will be more on the vulnerability exploitation part of the Schedule events feature of the Mango system. The goal will be to detect vulnerabilities or potential vulnerabilities and to prepare a report of the results.

4.1. Manual of penetration testing

In this section, the configuration of the penetration testing will be explained. For that test, the Mango system has to be configured to run properly without error. The first step is to download and run ZAP. Next, the configuration of the proxy in your browser to allow ZAP to scan your application is required. In that step, Firefox must be warned to allow ZAP to detect your browser search properly. After this configuration, the start of the DVWA is required and the address has to be set to localhost on the 127.0.0.1 IP address. ZAP also has to be set on a different port than the Mango system which is on 8080, it is possible to use port 8081 for the proxy. Now it is possible to select the type of vulnerability attack and proceed. For this experimentation, the quick attack is used to get all the types of vulnerability. The focus was directed on our feature functionality which is the Schedule events.

Configuration for ZAP:

URL to attack:	http://localhost:8080/test/scheduled_events.shtm
Proxy port	8081
Browser used:	Firefox Version 106.0.5 (64-bit)
Operating system	Ubuntu
User-Agent from ZAP	Mozilla/5.0 (Windows NT 10.0; WOW64; x64; rv:105.0esr) Gecko/20010101 Firefox/105.0esr

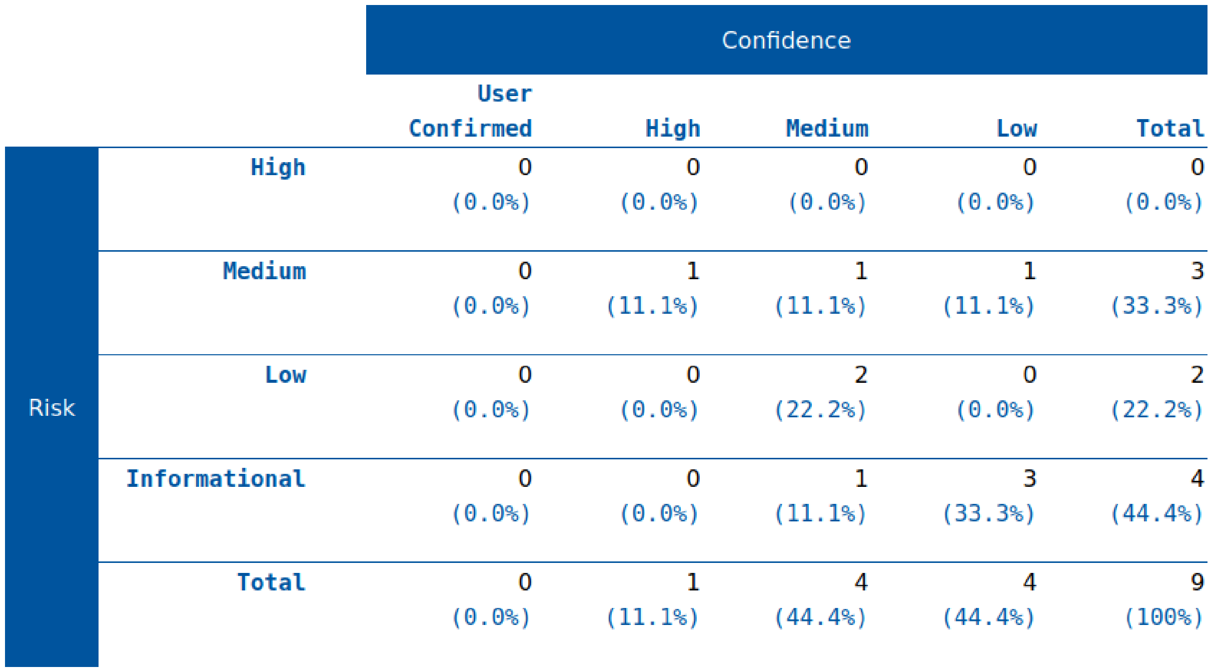
4.2. Testing result

In this section, a summary of the result of our penetration testing will be exposed and after each of the eight alerts, a short description of the vulnerability and its potential risk and details will be discussed.

4.2.1. Summary of the result

Alert counts by risk and confidence

This table shows the number of alerts for each level of risk and confidence included in the report.



		Confidence				
		User Confirmed	High	Medium	Low	Total
Risk	High	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	Medium	0 (0.0%)	1 (11.1%)	1 (11.1%)	1 (11.1%)	3 (33.3%)
	Low	0 (0.0%)	0 (0.0%)	2 (22.2%)	0 (0.0%)	2 (22.2%)
	Informational	0 (0.0%)	0 (0.0%)	1 (11.1%)	3 (33.3%)	4 (44.4%)
	Total	0 (0.0%)	1 (11.1%)	4 (44.4%)	4 (44.4%)	9 (100%)

Figure 4.2.1: summary of alert counts by risk and confidence

Alert counts by site and risk

This table shows, for each site for which one or more alerts were raised, the number of alerts raised at each risk level.

		Risk			
		High (= High)	Medium (>= Medium)	Low (>= Low)	Informational (>= Informational)
Site	http://localhost:8080	0 (0)	3 (3)	2 (5)	4 (9)

Figure 4.2.2: summary of alert counts by site and risk

Alert counts by alert type

This table shows the number of alerts of each alert type, together with the alert type's risk level.

Alert type	Risk	Count
Absence of Anti-CSRF Tokens	Medium	3 (33.3%)
Content Security Policy (CSP) Header Not Set	Medium	5 (55.6%)
Missing Anti-clickjacking Header	Medium	3 (33.3%)
Cookie without SameSite Attribute	Low	1 (11.1%)
X-Content-Type-Options Header Missing	Low	18 (200.0%)
Information Disclosure - Suspicious Comments	Informational	26 (288.9%)
Loosely Scoped Cookie	Informational	2 (22.2%)
Modern Web Application	Informational	4 (44.4%)
User Controllable HTML Element Attribute (Potential XSS)	Informational	2 (22.2%)
Total		9

Figure 4.2.1.3: summary of alert counts by type

4.2.2. Review of eight alerts

Here is a table for the summary of all eight alerts from the highest to the lowest severity level with the type of the alert. The type is only based on OWASP Top 10 - 2017 The Ten Most Critical Web Application Security Risks document [3]. Confidence defines the confidence that ZAP has about this alert.

Table 4.2.2: list of all the alerts by the highest to the lowest severity level

Alert number	Alert name	Type [3]	Type Description	CWE ID	WASC ID	Risk	Confidence
#1	Absence of Anti-CSRF Tokens	A5	Broken Access Control	352	9	Medium	Low
#2	Content Security Policy (CSP) Header Not Set	A6	Security Misconfiguration	693	15	Medium	High
#3	Missing Anti-clickjacking Header	A6	Security Misconfiguration	1021	15	Medium	Medium
#4	Cookie without SameSite Attribute	A5	Broken Access Control	1275	13	Low	Medium
#5	X-Content-Type-Options Header Missing	A6	Security Misconfiguration	693	15	Low	Medium
#6	Information Disclosure - Suspicious Comments	A3	Sensitive Data Exposure	200	13	Informational	Low
#7	Loosely Scoped Cookie	A6	Security Misconfiguration	565	15	Informational	Low
#8	User Controllable HTML Element Attribute (Potential XSS)	A1	Data injection attacks	20	20	Informational	Low

It is possible to see that, the penetration tests have identified a vulnerability of Data injection attack (A1), Sensitive Data Exposure (A3), Broken Access Control (A5) and Security Misconfiguration (A6). Those types are well described by OWASP documentation [4]. These risk codes are as listed below: RED means the highest risk, ORANGE means medium risk, YELLOW means low risk, BLUE means informative, and GREEN means it may be a false positive.

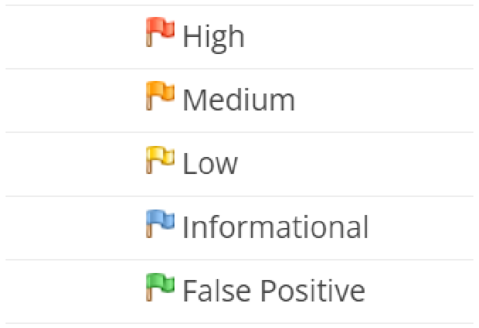


Figure 4.2.2: flags indicator of risk by ZAP

See details of alerts analysis in the appendix.

5. Compare static analysis and penetration testing results

Here is a breakdown of which OWASP type of error was detected by static analysis and which were detected by penetration testing.

Table 5.1: Comparison of OWASP categories detected by static code analysis and penetration testing

OWASP category	Static Code Analysis	Penetration Testing
A1 - Injection	✓	✓
A3 - Sensitive Data Exposure	✓	✓
A5 - Broken Access Control		✓
A6 - Security Misconfiguration	✓	✓
A7 - Cross-Site Scripting	✓	

In this table, 2 differences can be noted: First, only penetration testing detected a "Broken Access Control" vulnerability, denoting that users can perform unauthorized actions. Also, only static code analysis detected a "Cross Site Scripting" vulnerability, which has to do with exploitable code.

Since penetration testing is a black box testing method that performs automated tests using the application's front end without the knowledge of the source code, it should tend to expose vulnerabilities that have to do with data inputs, cookies, and requests for example. Indeed, the vulnerabilities with the CWE ID 565 and 1275 have to do with improperly validated or improperly configured cookies, whereas 20, 693, and 1021 have to do with the HTML that was served to the client. Finally, 352 signals that malicious requests could be accepted, while 200 signals that sensitive information could be leaked to a malicious actor [6]. As expected, most vulnerabilities are linked to potential concrete attacks.

In comparison, static code analysis is white box testing and therefore has access to the source code. Thus, the vulnerabilities found by static code analysis should be closely related to the application's source code. For instance, the vulnerability with the CWE ID 94 signals that some code is dependent on external inputs, and could therefore be exploited. 338, 1333 and 319 respectively identify a weak random number generator, inefficient regular expressions, and communication of sensitive information in the clear text [6]. In short, except for the vulnerability with the CWE ID 319, it is obvious that all vulnerabilities could only be found with access to the source code.

Also, static code analysis produced 297 security hotspots, compared to a total count of 64 vulnerabilities for penetration testing. Such a high count for static code analysis might be a sign that it produces more false positives, although manual review would be required to confirm this conjecture. On the other hand, penetration testing would have fewer false positives because the analysis is done from the point of view of an attacker, and is, therefore, closer to a real-world scenario.

6. Conclusion

In this assignment, we learned how to perform static analysis using sonarCloud on the entire Mango system. Two vulnerabilities and a lot of security hotspots were identified with this tool. Six hotspots and two vulnerabilities were analyzed and some recommendations on how to fix them were given.

We also learned how to perform manual penetration tests using the ZAP tool on the schedule events feature specifically. We tracked alert counts by risk and confidence, site and risk and finally by alert type. Then we organized the 8 alerts from highest to lowest on the severity level by the type of alert.

In the end, a thorough comparison of the results obtained with static analysis versus penetration testing was presented. In short, penetration testing focuses on vulnerabilities coming from data inputs, cookies and request call since it is based on black box testing out of the end user's interface. On the other hand, static code analysis is more suitable to detect vulnerabilities closely linked to the application's source code. To conclude, while static code analysis is more thorough and leads to more vulnerabilities detected by sheer number compared to penetration testing, it also leads to more false positives.

7. References

[1] GitHub. (n.d.). Docs.sonarcloud.io. Retrieved November 30, 2022, from https://docs.sonarcloud.io/getting-started/github/

[2] *Owasp vulnerable web applications directory*. OWASP Vulnerable Web Applications Directory | OWASP Foundation. (n.d.). Retrieved November 22, 2022, from https://owasp.org/www-project-vulnerable-web-applications-directory/

[3] 2017 Top 10 | OWASP. (n.d.). Owasp.org. ttps://owasp.org/www-project-top-ten/2017/Top_10

[4] OWASP ZAP – Alerts. (n.d.). Www.zaproxy.org. Retrieved November 29, 2022, from https://www.zaproxy.org/docs/desktop/start/features/alerts/

[5] Static analysis. Retrieved November 30, 2022, from https://sonarcloud.io/summary/overall?id=KyrollosBekhet_SQE-TA-Fall-2022

[6] *Common weakness enumeration*. CWE. (n.d.). Retrieved November 28, 2022, from https://cwe.mitre.org/

8. Appendix

Alerts details:

Name	#1 - Absence of Anti-CSRF Tokens
URL	GET http://localhost:8080/test/scheduled_events.shtm
Risk	Medium
Confidence	Low
Short description	No Anti-CSRF tokens were found in an HTML submission form. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. This issue means that the HTML form is not protected by any type of token.

Potential risk	The absence of Anti-CSRF tokens may lead to a Cross-Site Request Forgery or cross-site scripting (XSS) exploits that can result in executing a specific application action as another logged-in user and stealing their account user and password or other sensitive information.
Solution	This problem can be resolved by adding a temporarily valid token to any form submission in your application. Recommendation from Common Weakness Enumeration (CWE) lists the for this alert. 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 defences 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 Referer for privacy reasons. Reference http://cwe.mitre.org/data/definitions/352.html

Name	#2 - Content Security Policy (CSP) Header Not Set
URL	GET http://localhost:8080/test/scheduled_events.shtm
Risk	Medium

Confidence	High
Short description	Content Security Policy (CSP) means that configuration policies for the return object if not well configure and easily accessible in the HTTP Header. Content Security Policy (CSP) adds a layer of security which helps to detect and mitigate certain types of attacks such as Cross-Site Scripting (XSS) and data injection attacks. Hackers use XSS attacks to trick trusted websites into delivering malicious content. The browser executes all code from trusted origin and can't differentiate between legitimate and malicious code, so any injected code is executed as well.
Potential risk	The types of attacks that can be used here is Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware.
Solution	What can be a good solution is you need to configure your web server to return the Content-Security-Policy HTTP Header and give it values to control what resources the browser is allowed to load for your page. Recommendation from Common Weakness Enumeration (CWE) lists the for this alert. 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+. References https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy https://content-security-policy.com/ https://caniuse.com/contentsecuritypolicy https://web.dev/csp/

Name	#3 - Missing Anti-clickjacking Header
URL	GET http://localhost:8080/test/scheduled_events.shtm
Risk	Medium
Confidence	Medium

Short description	The response does not include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options to protect against 'ClickJacking' attacks.
Potential risk	Could be the same as Alert #2, such as Cross-Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware.
Solution	Recommendation from Common Weakness Enumeration (CWE) lists the for this alert. 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. Reference https://cwe.mitre.org/data/definitions/1021.html

Name	#4 - Cookie without SameSite Attribute
URL	GET http://localhost:8080/test/scheduled_events.shtm
Risk	Medium
Confidence	High
Short description	A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a 'cross-site' request. The SameSite attribute is an effective countermeasure to cross-site request forgery, cross-site script inclusion, and timing attacks.
Potential risk	One type of attack is the cookie can be sent as a result of a 'cross-site' request. Also, cross-site request forgery, cross-site script inclusion, and timing attacks can be used in a missing Samesite attribute.
Solution	Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies. Reference

	https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site

Name	#5 - X-Content-Type-Options Header Missing
URL	GET http://localhost:8080/test/login.htm
Risk	Low
Confidence	Medium
Short description	The missing "X-Content-Type-Options" header enables a browser to perform MIME type sniffing when the Content-Type header is not set or its value seems inappropriate.
Potential risk	Can lead to serious security issues with an XSS attack. In an application that allows an upload of jpg files, an attacker may upload a file with a jpg extension being in fact an HTML file with malicious JavaScript inside. If the MIME type sniffing is enabled, the browser handles the file as HTML and executes the file.
Solution	Ensure that the application/web server sets the Content-Type header appropriately and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. X-Content-Type-Options = nosniff If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing. References https://scanrepeat.com/web-security-knowledge-base/x-frame-options-header-not-set https://owasp.org/www-community/Security_Headers

Name	#6 - Information Disclosure - Suspicious Comments
URL	GET http://localhost:8080/test/scheduled_events.shtm
Risk	Informational
Confidence	Low
Short description	Information disclosure is when a web application accidentally reveals sensitive information to users.
Potential risk	The response appears to contain suspicious comments which may help an attacker. This information can be user data, business data, or technical details about the application and its infrastructure.
Solution	Remove all comments from the application that return information that may help an attacker.

Name	#7 - Loosely Scoped Cookie
URL	GET http://localhost:8080/test/scheduled_events.shtm
Risk	Informational
Confidence	Low
Short description	The domain scope applied to a cookie determines which domains can access it. For example, a cookie can be scoped strictly to a subdomain e.g. www.mywebsite.com, or loosely scoped to a parent domain e.g. mywebsite.com. In the latter case, any subdomain of mywebsite.com can access the cookie.
Potential risk	A attacker can use this loosely scoped to use a cookie to a subdomain of a website and access more data and the web page of the application.

Solution	Always scopes cookies to an FQDN (Fully Qualified Domain Name). In that way, you can protect the subdomain of your website. References https://tools.ietf.org/html/rfc6265#section-4.1 https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/06-Session_Management_Testing/02-Testing_for_Cookies_Attributes.html http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy_for_cookies

Name	#8 - User Controllable HTML Element Attribute (Potential XSS)
URL	GET http://localhost:8080/test/login.htm
Risk	Informational
Confidence	Low
Short description	A user controllable HTML Element Attribute is the concept that an attacker can look at use input in a query parameter, for example, a POST data to be able to read and controlled them.
Potential risk	This check looks at user-supplied input in query string parameters and POST data to identify where certain HTML attribute values might be controlled. This provides hot-spot detection for XSS (cross-site scripting) that will require further review by a security analyst to determine exploitability.
Solution	Validate all input and sanitize output before writing to any HTML attributes. Reference http://websecuritytool.codeplex.com/wikipage?title=Checks#user-controlled-html-attribute