École Polytechnique de Montréal

POLYTECHNIQUE MONTRÉAL

LOG8371E: Software Quality Engineering

Third travaux pratique (TP3)

Professor: Rhouma Naceur

Team members

Student ID	Names
2313663	Xi-Zhen Wang
2246556	Abid Ullah Khan
2316903	Shitian Li
2313674	Fábio Akira Yonamine
2317239	Roberto Molina

Montréal, Quebec
2023

Table of Contents

I. Abstract

This study presents a comprehensive evaluation of the PetClinic system's security using static analysis via SonarQube and penetration testing with OWASP ZAP. The investigation aimed to pinpoint vulnerabilities in the REST controller, focusing on security pitfalls and weaknesses. Static analysis revealed eight vulnerabilities across various severity levels, categorized by OWASP, SANS, or CWE standards. Additionally, penetration testing identified eight security hotspots, highlighting attack scenarios and offering actionable recommendations for resolution. Challenges in aligning identified vulnerabilities between the two methods underscored nuanced detection methodologies. The comprehensive analysis equips stakeholders with insights to fortify the PetClinic system against potential vulnerabilities, fostering an in-depth understanding of risks and their mitigations.

II. Introduction

Our exploration of the PetClinic system involved a comprehensive investigation of the code using both static analysis via SonarCloud/SonarQube and penetrating testing utilizing OWASP ZAP. The aim was to unearth vulnerabilities within the system's REST controller and the overall system, recognizing potential security pitfalls and weaknesses. Throughout this process, detailed static analysis was executed, and the findings culminated in a comprehensive report that not only offers a summarized overview but also delves deeply into eight identified vulnerabilities or security hotspots, spanning Blocker, Critical, and Major classifications. These issues were noted across various types, enabling us to present a vivid picture of the risks. Each vulnerability comes with a description, potential risks illustrated through attack scenarios, file identification, severity level, and vulnerability classification based on recognized standards such as OWASP, SANS, or CWE. To fortify the system, actionable recommendations for each vulnerability are provided.

Simultaneously, we executed through penetration testing, recording our observations meticulously. The resulting report encapsulates eight security hotspots or issues categorized by severity levels and distinct vulnerability types. Our investigation revealed potential vulnerabilities and their associated risks, mapped with attack scenarios, URL identification, severity levels, and classifications aligning with OWASP or CWE standards. For each, we devised actionable recommendations for resolution. Our approach encompassed a step-by-step manual detailing the execution of both static analysis and penetration testing. Throughout this process, we encountered challenges that required creative solutions, and we detailed these obstacles and our effective resolutions. Central to our investigation is the comparison between the results derived from static analysis and penetration testing. We noticed discrepancies in the numbers and categories of Common Weakness Enumeration (CWE) identified by each method. This comparison allowed us to unravel the nuances behind why certain vulnerability categories were exclusively detected by one approach over the other, enhancing our understanding of the system's security landscape.

In presenting this comprehensive analysis, our aim is to provide actionable insights to fortify the PetClinic system, empowering stakeholders with a detailed understanding of potential vulnerabilities and their mitigations.

III. Static Analysis

To run the static analysis, we chose to use SonarQube, because we have already used it in TP1. In order to not have a lot of issues installing and running it, we decided to install it indirectly through Docker, as instructed in the SonarQube tutorial, using the command:

```
docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
```

After that command is finished with the installation and configuration, we were able to finally use the tool, accessing it locally at the following address:

```
localhost:9000/
```

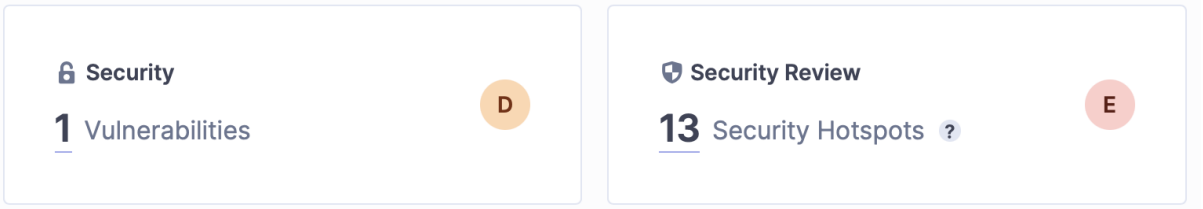
To avoid any complicated installations and set-ups to use SonarScanner, we decided to use Maven instead, as it is also compatible with Java projects and usually, it's something already installed on some computers. In case you don't have it installed locally yet, you might be able to do so by running the following command on MacOS, using the Homebrew tool (for Windows or Linux, the procedure for installing it might differ):

```
brew install maven
```

After installing Maven and SonarQube, we created a new Local Project, with the same project display name and project key, with "main" as main branch name. To finally run the analysis, using Maven, on a Terminal screen, we go to the same folder as the PetClinic code is located at, and we run the following command, as stated in the SonarQube page:

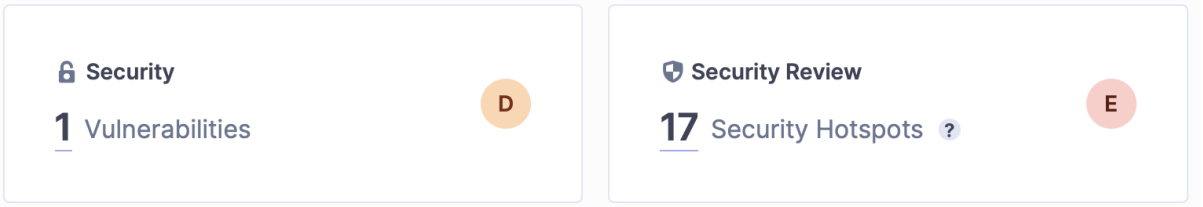
```
mvn clean verify sonar:sonar \
  -Dsonar.projectKey=tp3analysis \
  -Dsonar.projectName='tp3analysis' \
  -Dsonar.host.url=http://localhost:9000 \
  -Dsonar.token=sqp_92e562053097a1c96725b9fb2d1769ce64599c60
```

After some time, SonarQube should run the analysis and present the results concerning security issues, among other analyses previously used in TP1. As results for the PetClinic analysis, we got the following numbers:



By running the SonarQube analysis on the original code available in the GitHub repository, we realized that we still didn't have eight different vulnerability or security hotspots required on this TP3. Because of that, as instructed by the teacher's assistant, we artificially introduced new security hotspots inside the original code, in order to meet the eight different ones, by consulting new security issues on the SonarSource Rules [1] page. This was the only difficulty that we actually had in the static analysis task, before getting to the results' analysis, reading the documentation on the security vulnerabilities and writing this report.

The final result shown to us by SonarQube is as it follows:



A. Vulnerabilities and Security Hotspots

1. Allowing safe and unsafe HTTP methods is security-sensitive

Type: Cross-Site Request Forgery (CSRF) (CWE-352)
Tags: cwe spring
File: RootRestController.java
Severity Level: Minor
OWASP 2021 A1 Type: Broken Access Control

The vulnerability is identified within the application on line 41 of the RootRestController.java file, where the redirectToSwagger function permits unrestricted access by allowing both safe (GET, HEAD, and OPTIONS) and unsafe (POST, PUT, and DELETE) requests. This lack of method specification poses a potential risk, as it opens the door to unintended and potentially harmful interactions.

Allowing a combination of safe and unsafe HTTP methods for specific operations in a web application could compromise its security, particularly considering that CSRF protections are often designed to safeguard against operations performed by unsafe HTTP methods.

The solution to this issue is to define the redirectToSwagger endpoint as a GET method. So, it will be a safe method as it does not permit the modification of the system.

The issue has been cataloged as a broken access control in OWASP 2021 A1, this means that it allows users to act outside of their intended permissions.

2. Disabling Spring Security's CSRF protections is security-sensitive

Type: Cross-Site Request Forgery (CSRF) (CWE-352)
Tags: cwe spring
File: BasicAuthenticationConfig.java
Severity Level: Critical
OWASP 2021 A1 Type: Broken Access Control

This issue is because in line 35 the CSRF protection is disabled. The CSRF (Cross-Site Request Forgery) protections play a crucial role in ensuring the security of a web application, and intentionally turning off or disabling these protections can introduce significant security risks.

CSRF is a type of attack where an attacker tricks a user's browser into making an unintended request on a web application in which the user is authenticated. If CSRF protections are disabled, it means that the application is not implementing countermeasures to prevent or mitigate CSRF attacks.

CSRF protections typically involve mechanisms like anti-CSRF tokens, which are unique tokens generated and validated by the server to ensure that a particular request originates from a legitimate and authenticated user, not from a malicious source trying to exploit the user's credentials.

The resolution for this problem involves removing lines 34 and 35, which disable CSRF protection. By eliminating these lines, CSRF protection is retained, preventing the success of this type of attack.

It has been categorized as a Broken Access Control under the OWASP 2021 A1 classification. This is attributed to the fact that disabling CSRF protection can potentially allow attackers to gain unauthorized access to areas of the system where such access should be restricted.

3. Having a permissive Cross-Origin Resource Sharing policy is security-sensitive

Type: Insecure Configuration (CWE-346, CWE-942)
Tags: cwe spring
File: VisitRestController.java
Severity Level: Minor
OWASP 2021 A5 Type: Security Misconfiguration

In line 48 the headers errors and content-type are exposed. This indicates that the server is allowing the client to access these headers in the response. While exposing headers is a common practice, doing so without proper consideration can lead to unintended security consequences.

This Security hotspot means that allowing broad and unrestricted access to resources from different origins can pose security risks. CORS is a security feature implemented by web browsers to control how web pages in one domain can request and interact with resources hosted on another domain.

When a CORS policy is permissive, it means that there are fewer restrictions on which origins are allowed to access the resources. This openness can potentially be exploited by malicious actors, leading to several types of attacks like CVE-2018-0269 or CVE-2017-14460.

One solution to this issue will be to restrict the domains that are allowed to see these headers. So, the other domains can not access this information.

The OWASP 2021 named this issue as a Security misconfiguration as having a permissive CORS is a bad configuration of the server.

4. Delivering code in production with debug features activated is security-sensitive

Type: Insecure Configuration (CWE-489, CWE-215)
Tags: cwe error-handling spring debug user-experience
File: ExceptionControllerAdvice.java
Severity Level: Minor
OWASP 2021 A5 Type: Security Misconfiguration

This security hotspot appears in line 50 as is where a debug instruction is being used. This instruction can give valuable information when the exception is thrown. This information can be very helpful during the debugging phase but in production can give very really sensitive data like the application's path or file names.

The solution to fix this security hotspot is to EnableWebSecurity annotation for SpringFramework with debug to false. This instruction will deactivate the debugging support making that code like the one on line 50 does not execute.

The OWASP classified this issue as a security misconfiguration as it is a bad practice of the developer to leave this code in production making it a bad configuration of the security of the system.

5. Hard-coded passwords are security-sensitive

Type: Authentication (CWE-798, CWE-259)
Tags: cwe cert
File: UserRestController.java
Severity Level: Blocker
OWASP 2021 A7 Type: Identification and Authentication failures

This security hotspot appears in line 57 as is where a password was registered directly in the code, without any kind of encryption. Because it's easy to extract any strings from an application source code

or binary, this makes the whole application at risk if there's sensitive data being protected by that password.

The solution to fix this security hotspot is to store the credentials in a separate file that's not pushed to the repository. There's also the possibility of storing these in a database or using cloud services to store passwords and credentials. In case a password has been previously exposed this way, the administrator should also change it.

The OWASP classified this issue as an identification and authentication failure, because exposing the password can lead to unauthorized access to code and sensitive data inside databases..

6. Using pseudorandom number generators (PRNGs) is security-sensitive

Type: Weak cryptography (CWE-338, CWE-330, CWE-326, CWE-1241)
Tags: cwe
File: PetRestController.java
Severity Level: Critical
OWASP 2021 A2 Type: Cryptographic failures

This security hotspot appears in line 57, as is where there's an instruction to generate a pseudorandom number. This instruction can leave the system prone to attacks, as the next generated numbers might be guessed because the algorithm used is not entirely random and safe, leading ultimately to an user being impersonated or accessing sensitive data.

The solution to fix this security hotspot is to exchange the Random() method for the SecureRandom() which is a cryptographically strong random number generator and, thus, generates unguessable numbers. Other fixes would be to use the generated numbers only once, to not expose them and, if there's the need of storing them, do so in secure files and/or databases.

The OWASP classified this issue as a cryptographic failure because, even though there's the implementation of a cryptography method for protecting data, it can be easily breached by malicious users.

7. Using unsafe Jackson deserialization configuration is security-sensitive

Type: Object Injection (CWE-502)
Tags: cwe
File: VisitRestController.java
Severity Level: Critical
OWASP 2021 A8 Type: Software and Data Integrity Failures

This security hotspot appears in line 108 as is where Jackson is configured to allow Polymorphic Type Handling (PTH). This is usually a security concern because it may allow an attacker to perform remote code execution when using "deserialization gadgets".

The solution to fix this security hotspot is to use the latest patch versions of jackson-databind, which include fixes to the already discovered "deserialization gadgets". Also, it's important to avoid using the default typing configuration, by using ObjectMapper.enableDefaultTyping().

The OWASP classified this issue as a software and data integrity failure as it may allow the system to run malicious code that changes the code itself or the databases, compromising its integrity.

8. Using hard coded IP addresses is security-sensitive

Type: Others (No CWE code reported)
Tags: cert
File: UserRestController.java
Severity Level: Minor
OWASP 2021 A1 Type: Broken access control

This security hotspot appears in line 62 as is where a hard-coded IP address is being used. This line can allow hackers to easily get a potentially sensitive address, perhaps facilitating DDoS attacks, getting access to systems or spoofing that address to bypass security checks. Also, because the IP address is hard-coded, it makes maintainability harder, because every time that address changes, a team has to manually alter it to the newest one in use.

The solution to fix this security hotspot is to simply make it configurable through environment variables, configuration files or any other similar approach.

The OWASP classified this issue as a broken access control, as hard-coded IPs could ultimately lead to unauthorized users performing harmful actions when, for example, executing DDoS attacks and getting unwanted access to a database storing sensitive information, for example.

Cross-Site Request Forgery (CSRF)
Insecure Configuration
Authentication
Cryptography

Static analysis: 45/45

IV. Penetration Test

A. Summary of Results

In this section, we use the OWASP ZAP tool to perform penetration testing of the entire PetClinic system. After using spider and actively scanning the PetClinic system, we obtained an overall report. The following data are based on the report result.

		Confidence				
		User confirmed	**High**	**Medium**	**Low**	**Total**
Risk	**High**	0	0	0	1 (11.1%)	1 (11.1%)
	Medium	0	1 (11.1%)	1(11.1%)	0	2 (22.2%)
	Low	0	0	0	2 (22.2%)	2 (22.2%)
	Informational	0	0	2 (22.2%)	2 (22.2%)	4 (44.4%)
	Total	0	1 (11.1%)	3 (33.3%)	5 (55.6%)	9 (100%)

Table X: Alert counts by risk and confidence

Confidence: In penetration testing, confidence refers to how confident the testing tool is that it has discovered a vulnerability.

From the table above, we can observe that the PetClinic system has a total of 9 vulnerabilities, of which 22.2% are at low risk level, 22.2% are at medium risk level, 11.1% are at high risk level, and 44.4% are message notification level. It is worth noting that there are 4 Information risks described in our report. However, Information risks are not real vulnerabilities but prompts about system information provided by penetration testing tools. This type of tip is general information about the system, not potential security risks.

Consequently, there are a total of 9 vulnerabilities in the system. Four of the medium-confidence vulnerabilities can be exempt from manual inspection again because the system's credibility is relatively high. But the remaining five low-confidence vulnerabilities may require manual verification of the existence of the vulnerability, especially if it contains a high-risk vulnerability.

		Risk			
		Information	**High**	**Medium**	**Low**
Site	**http://localhost:9966**	4	1	2	2

Table X: Alert counts by risk and site

We run the PetClinic system locally on port 9966, and then connect ZAP to that port, allowing ZAP to act as a middleman for testing. Therefore, what is shown in the report is http://localhost:9966. From the above table, we can see that the data result is similar to Figure X, which can prove our result is correct.

Alert Type	Risk	Count
Path Traversal	High	1 (11.1%)
Content Security Policy (CSP) Header Not Set	Medium	6 (66.7%)
Spring Actuator Information Leak	Medium	1 (11.1%)
Timestamp Disclosure- Unix	Low	74 (822.2%)
Cross-Site Scripting Weakness (Persistent in JSON Response)	Low	2 (22.2%)
Information Disclosure - Suspicious Comments	Information	3 (33.3%)
Modern Web Applications	Information	2 (22.2%)
User Agent Fuzzer	Information	276 (3,066.7%)
Authentication Request Identified	Information	1 (11.1%)

Table X: Alert counts by alert type

In the table above, we can see what the nine vulnerabilities we mentioned earlier are. Among them, as the path traversal with the highest risk, there is only one. This could lead an attacker to cross the application's restricted directory and gain access to sensitive information. While in the Petclinic system, we found 6 medium-risk vulnerabilities indicating that the CSP header is not set. This can make the application more vulnerable to malicious script injection. Another medium-risk vulnerability, involving Spring Actuator information disclosure, allows an attacker to gain access to sensitive system information through this vulnerability. And among the low-risk vulnerabilities, we found 74 involving Unix timestamp leaks and 2 low-risk vulnerabilities involving cross-site scripting that persists in JSON responses. Although they are low risk, they may leak some sensitive information about the system while enabling an attacker to inject malicious scripts into the application. As for Information Disclosure - Suspicious Comments Information, Modern Web Applications Information, User Agent Fuzze, and Authentication request finalized, they all belong to the Information notation category, with a total of 282. Although they may reveal some information about the application structure and implementation, information notations are not vulnerabilities.

B. Comments for Alerts

In this section, we will analyze alert types from high to low risk and provide information about the severity level, vulnerability URL, and vulnerability type defined by OWASP. Furthermore, we also give recommendations about how to resolve these issues.

1. Path Traversal

Severity level: High
URL: http://localhost:9966/petclinic/api/owners?lastName=owners
OWASP_2021_A01: Broken Access Control

Path Traversal, also known as Directory Traversal or dot-dot-slash (../../) attack, is a vulnerability that occurs when an application allows attackers to execute or reveal the contents of arbitrary files located outside the web document root by manipulating the URL. This type of attack can potentially give unauthorized access to sensitive files or directories on a web server.

Suppose there is a web application that serves files based on user input, such as a file download feature. If the application does not properly validate user inputs, attackers could manipulate the input to navigate to directories outside of the intended scope. For example, users can download the files by using '*http://some_domain.com/get-files?file=report.pdf*', it is possible to insert a malicious string, such as '*?file=/etc/passwd*', as the parameter to access files located outside the web publish directory. In this case, the attacker can get the essential information required during login.

This alert type belongs to OWASP_2021_A01 - Broken Access Control, which represents it is the top 1 security risk in 2021. The relationship between Path Traversal and Broken Access Control lies in the context of access control vulnerabilities that can be exploited through path traversal attacks.

To prevent path traversal attacks, it's essential to **validate user inputs** and reject any input that deviates, especially when dealing with file paths. We can also use **web application firewalls (WAF)** to set the rules to detect and block path traversal attempts.

2. Content Security Policy (CSP) Header Not Set

Severity level: Medium
URLs: http://localhost:9966/petclinic/swagger-ui/index.html
http://localhost:9966/petclinic
http://localhost:9966/robots.txt
http://localhost:9966/sitemap.xml
http://localhost:9966/petclinic/swagger-ui/oauth2-redirect.html
OWASP_2021_A05: Security Misconfiguration

The application is susceptible to a vulnerability due to the absence of the Content-Security-Policy (CSP) header. CSP serves as an additional security layer designed to identify and mitigate specific types of attacks, including cross-site scripting (XSS) and data injection attacks.

One common attack that CSP helps prevent is Cross-Site Scripting (XSS). Without a CSP header, an attacker may inject malicious scripts into a website, and these scripts could then be executed in the context of other users' browsers, leading to unauthorized access, data theft, site defacement, or the distribution of malware through XSS attacks.

This alert type belongs to OWASP_2021_A05 - Security Misconfiguration which refers to the improper implementation or configuration of security controls, leading to vulnerabilities that attackers can exploit. Therefore, "Content Security Policy (CSP) Header Not Set" is one of the scenarios of Security Misconfiguration, the absence of proper CSP headers is considered a security misconfiguration.

To address this issue, developers should implement and configure a **Content-Security-Policy** in HTTP response header from the web server.

3. Spring Actuator Information Leak

Severity level: Medium
URL: http://localhost:9966/petclinic/actuator/health
OWASP_2021_A01: Broken Access Control

Spring Actuator is a set of features in the Spring Framework that help to monitor and manage the application. However, improper configuration or security settings can lead to information leaks, allowing unauthorized users to access sensitive details about the application, such as environment properties, configuration, and other internal information.

An attacker may exploit Spring Actuator information leaks to gather sensitive information about the application, such as database connection details or other configuration parameters. This information can be used to plan more targeted attacks or gain unauthorized access to the system.

This alert type also belongs to OWASP_2021_A01 - Broken Access Control. Spring Actuator Information Leak is obviously due to the incorrect spring configuration being set leading to potential security breaches.

To mitigate this risk, we recommend **disabling Health Actuators** and other Actuators by editing the configuration files (typically application.properties or application.yml) or **restricting their access** to administrative users only. This will help prevent information leakage and enhance the overall security of the application.

4. Timestamp Disclosure - Unix

Severity level: Low
URL: http://localhost:9966/petclinic/swagger-ui/swagger-ui-standalone-preset.js
http://localhost:9966/petclinic/swagger-ui/swagger-ui-bundle.js
OWASP_2021_A01: Broken Access Control

Timestamp Disclosure in Unix occurs when a web server provides detailed error messages that reveal information about the system's internal configuration, including the timestamp of the server. This information can be valuable for attackers attempting to identify potential weaknesses in the system.

For example, attackers might send a deliberately malformed request to a web server. If the server is misconfigured or vulnerable, it may respond with an error message that includes a timestamp. By analyzing these timestamps, an attacker can gain insights into the server's uptime, recent updates, and potentially identify outdated software versions. This information helps the attacker in planning and executing targeted attacks.

This alert type also belongs to OWASP_2021_A01 - Broken Access Control because Timestamp Disclosure is an example for exposure of sensitive system information, although a timestamp leak may appear harmless.

To solve this issue, we recommend **customizing the error handling** to provide generic information to users rather than using the error messages generated from the system. It can avoid exposing timestamps or sensitive information in error messages presented to users.

5. Cross-Site Scripting Weakness (Persistent in JSON Responses)

Severity level: Low
URL: http://localhost:9966/petclinic/api/owners
OWASP_2021_A03: Injection

Cross-Site Scripting (XSS) is a security vulnerability that allows attackers to inject malicious scripts into web applications. When it's persistent in JSON responses, which means the injected script is stored on the server and included in the JSON data sent to clients, leading to potential exploitation when the JSON is processed on the client side.

Here is a potential scenario where an attacker can use XSS to attack a system. There is a web application that allows users to submit comments, and these comments are stored in a JSON format on the server. An attacker could submit a comment containing malicious JavaScript code, such as: '*<script>alert(1);</script>*'. If this input is not properly sanitized on the server, the malicious script will be stored in the JSON response. When a user retrieves this JSON data and the application renders it in a way that interprets HTML and executes scripts (for example, using JavaScript's innerHTML), the attacker's script will be executed in the victim's browser context.

This alert type belongs to OWASP_2021_A03 - Injection, this attacks involve the malicious injection of untrusted data into a program or command. Attackers typically exploit vulnerabilities that allow them to inject and execute arbitrary code. Some common injections are SQL, OS command, and LDAP. We can see that XSS Weakness (Persistent in JSON Responses) is one of the injection aspects that come into play during the process of injecting malicious scripts into the web application.

To prevent Persistent XSS in JSON responses, we can also **implement input validation and sanitization**. Another practical way is using **safe JSON parsers**, which can decode JSON on the client side and do not automatically execute scripts.

C. User Manual for ZAP

First, we use '*./mvnw spring-boot:run*' to launch the PetClinic application that we intend to test, and we are certain that the PetClinic is running on the local port 9966. Then, in the ZAP software, we select Manual mode and input the target URL as "http://localhost:9966/petclinic" for testing. After ensuring everything is set up successfully, we initiate a spider on "http://localhost:9966" as a preliminary step in our penetration test. This step aims to establish a comprehensive understanding of the target application.

Finally, we perform an active scan on "http://localhost:9966" to simulate an attacker conducting a real attack on the target application. This process helps uncover potential vulnerabilities and security risks, which will be documented in the report we generate.

Penetration test: 45/45

V. Comparison

In this part we are gonna compare the differences between a static analysis and penetration testing. Static analysis (SonarQube) and penetration testing (ZAP) are two distinct approaches to evaluating the security of software systems.

To compare both approaches we are going to observe the differences between the security issues that both of them have identified. In order to do it we have observed each issue identified by the tools and their CWE category.

A. CWE categories found in SonarQube

Issue Name	CWE ID	CWE Name
Allowing safe and unsafe HTTP methods is security-sensitive	352	Cross-Site Request Forgery (CSRF)
Disabling Spring Security's CSRF protections is security-sensitive	352	Cross-Site Request Forgery (CSRF)
Having a permissive Cross-Origin Resource Sharing policy is security-sensitive	346	Origin Validation Error
	942	Permissive Cross-domain Policy with Untrusted Domains
Delivering code in production with debug features activated is security-sensitive	489	Active Debug Code
	215	Information Exposure Through Debug Information
Hard-coded passwords are security-sensitive	798	Use of hard-coded Credentials
	259	Use of hard-coded Password
Using pseudorandom number generators	338	Use of Cryptographically Weak

(PRNGs) is security-sensitive		Pseudo-Random Number Generator (PRNG)
	330	Use of Insufficiently Random Values
	326	Inadequate Encryption Strength
	1241	Use of Predictable Algorithm in Random Number Generator
Using unsafe Jackson deserialization configuration is security-sensitive	502	Deserialization of Untrusted Data
Using hard coded IP addresses is security-sensitive	-	-

B. CWE categories found in ZAP

Issue Name	CWE ID	CWE Name
Path Traversal	22	Improper Limitation of a Pathname to a Restricted Directory
Content Security Policy (CSP) Header Not Set	693	Protection Mechanism Failure
Spring Actuator Information Leak	215	Insertion of Sensitive Information Into Debugging Code
Cross Site Scripting Weakness (Persistent in JSON Response)	79	Improper Neutralization of Input During Web Page Generation
Timestamp Disclosure - Unix	200	Exposure of Sensitive Information to an Unauthorized Actor

C. Comparision

While both static analysis and penetration testing aim to identify and address security vulnerabilities, the CWE categories identified by each method highlight their distinct focuses.

In the static analysis results, common issues such as Cross-Site Request Forgery (CSRF), origin validation errors, and insecure configurations like permissive cross-domain policies are prominent. This suggests a concentration on code-level weaknesses and misconfigurations that could lead to potential exploits.

On the other hand, the penetration test results reveal a focus on runtime and operational security concerns, including improper limitation of directory paths, protection mechanism failures, and exposure of sensitive information during web page generation. These findings suggest a concern for issues that

may arise during the actual execution of the application, emphasizing scenarios where an attacker could actively exploit vulnerabilities in a live environment.

In summary, while the static analysis tends to emphasize code-centric vulnerabilities and misconfigurations, the penetration testing emphasizes on runtime issues and the effectiveness of protection mechanisms in real-world scenarios. To conclude, we want to remark that to obtain a comprehensive understanding of the application's security there is required the combination of both approaches.

Difference between two tests: 10/10

VI. Conclusion

Our thorough examination employing static analysis and penetration testing elucidated multiple vulnerabilities within the PetClinic system. The static analysis uncovered code-centric issues like CSRF and insecure configurations, while penetration testing emphasized runtime concerns such as directory limitations and information exposure during execution. The comparison of results highlighted the complementary nature of both approaches, emphasizing the importance of combining them for a holistic understanding of system security. The detailed report of vulnerabilities and recommended actions provide stakeholders with actionable insights to enhance the PetClinic system's resilience against potential threats, ensuring robust security measures.

VII. References

[1] https://docs.sonarsource.com/sonarqube/latest/user-guide/issues/
[2] https://www.docker.com/blog/containerizing-a-legendary-petclinic-app-built-with-spring-boot/
[3] https://wilsonmar.github.io/spring-petclinic/

Quality of work: 100
Quality of presentation: 100

Grade: 10/10