SENG 426

Software Quality Engineering

Summer 2021

University of Victoria

Delivery 5: Security Testing

August 2, 2021

David Bishop	V00884250
Amaan Makhani	V00883520
Ben Austin	V00892013
Nolan Kurylo	V00893175
Liam Franke	V00887604
Evan Roubekas	V00891470

Table of Contents

Threat Modelling

Threat Vulnerability Matrix

Threat Name	Feature(s)	Vulnerabilities	Mitigation Solutions	Verification Cases	Status
Social Engineering	login	Attacker gains elevated access through manipulating users	User training/awareness	Probe sample users regularly and randomly	Complete
Phishing	login	Open redirect, XSS< SQL injection, session splitting	Input validation, output encoding	Ensure site has no exploitable command injection (SQL, XSS, etc) vulnerability	Complete
Password Cracking	login	Weak password allowed easy access to login info/services	Multi Factor authentication, password strength checking, account lockout	Ensure that password format is checked proactively and lockout mechanism kiosks in after limited number of failed guesses	Complete
Leak Password Hashes	login	Directory traversal	Input validation	Check for exploitable DT vulnerability	Complete
Sniff & Replay	login	Passwords exchanged in cleartext or using weak cypher	Multi Factor authentication, use SSL to ensure data is not modified/viewed in transit	Use sniffer to capture and analyze traffic sample, ensure that underlying system meets standard practice	Complete
Session Hijacking	login	Predictable or recoverable session ID, session splitting, open redirect	Renew session ID, ensure random session ID of appropriate length, input validation	Collect session IDs over a number of sessions and examine distribution and length	Complete
Denial of Service (DOS)	login		Monitor system resources (network, CPU, memory, etc)	Checking presence and effectiveness of monitoring scheme	Complete
Rig Bid		XSS, SQL injection	Input validation, output encoding	Ensure that site has no exploitable command injection (SQL, XSS, etc) vulnerability	Complete

Bypass Registration	Register, account management	Directory traversal	Input validation	Check for exploitable DT vulnerability	Complete
Tap Communications		Communications in cleartext or over weak systems	Use SSL to ensure data is not modified/viewed in transit	Assert that communications cannot be established without the use of SSL	Complete

High-level Application Features

- Logging in to the account
- Registering for an account
- Transferring money
- Managing transactions
- Requesting an account
- Exchange crypto
- Exchange currency
- Managing customers
- Managing payees
- Transferring money to a payee
- Uploading Id proof
- Update username/password
- Managing users

Static Source Code Analysis

As part of our Jenkins pipeline, our code is sent to SonarQube for analysis every time the application is pushed to gitlab. For this deliverable, the latest changes from the previous deliverable were pushed to our master branch, and the resulting reports were generated by SonarQube. In SonarQube, issues marked under the *vulnerabilities* and *security hotspot* sections were individually analyzed to determine their impact on the application. Issues were marked according to whether they were a real threat, or a false alarm. After real issues were identified, issue type and severity were updated to reflect the issue and avoid mis-classifications by SonarQube. Appropriate commentary was given to all issues.

Issues and Solutions

In the initial sonarqube scan, there were 8 identified vulnerabilities and 89 security hotspot issues that required attention. After analyzing every security hotspot issue, the main

vulnerabilities were related to the storage and return of mutable members, the concern of permission handling, and the concern of proper use of regular expressions.

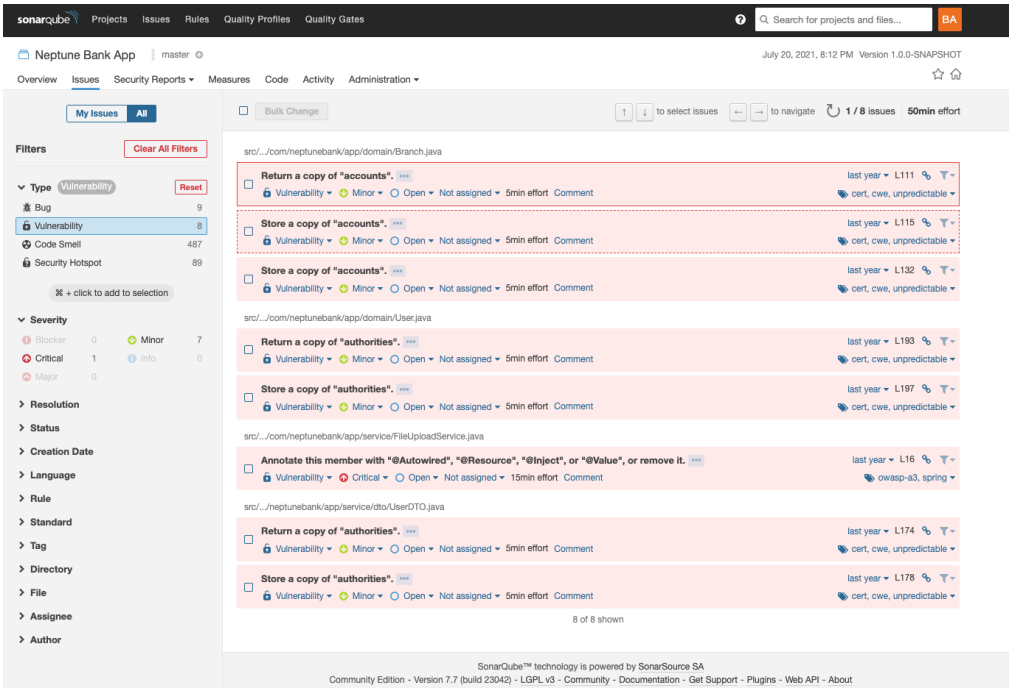
As for the storage and return of mutable members, in order to avoid the possibility of unexpected changes in the class states, the *Collections.unmodifiableSet()* wrapper was used on all setters / getters in question to prevent the internal state of the object and prevent it from being modified if an attacker were to attempt to manipulate its state maliciously.

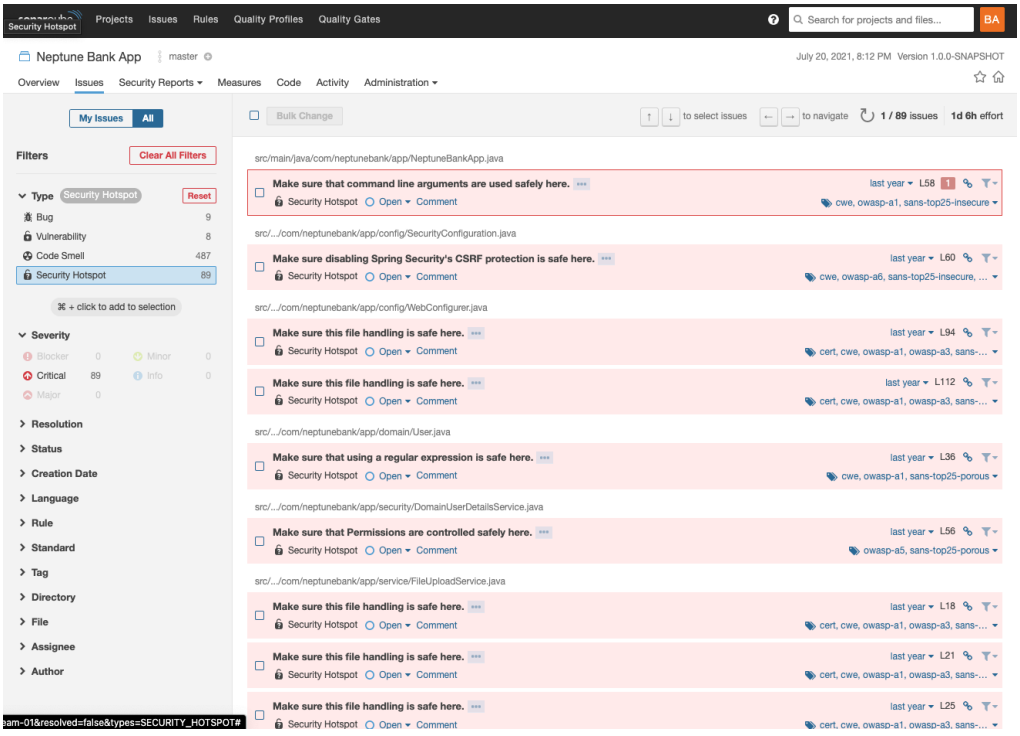
For permission handling vulnerabilities, the code was analyzed and determined to be safe, as there were already substantial checks and validation in place regarding user authority and roles throughout the original source code. ZAP can be used to determine whether this analysis was correct or not.

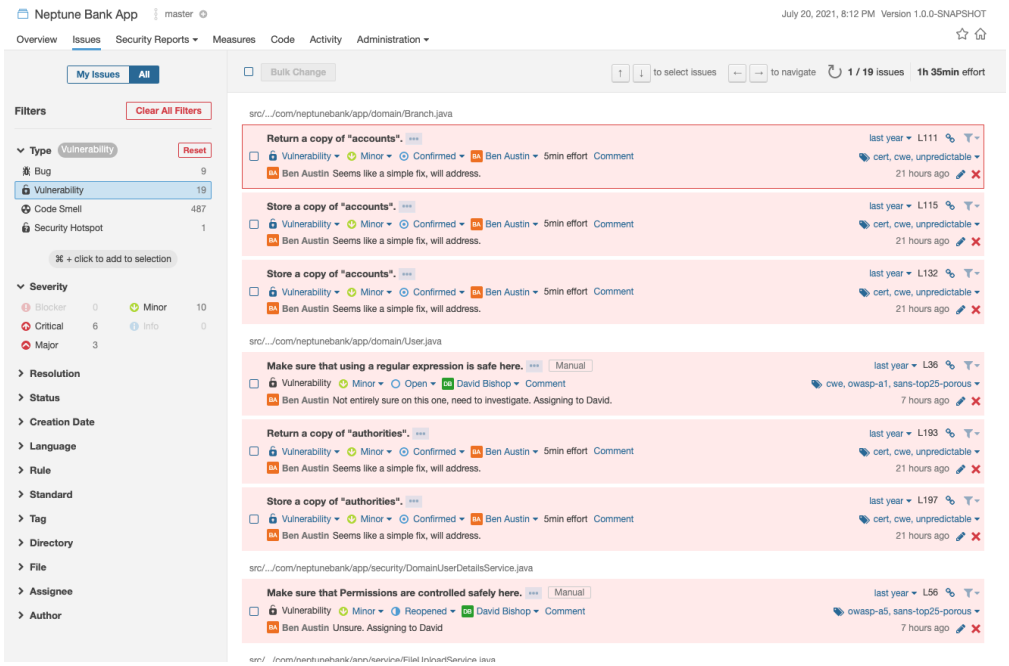
Finally, regarding the proper use of regular expressions, this can be more difficult to analyze than one would expect. “Evil” regex is a regular expression that can be stuck on crafted input. Regular expression Denial of Service (ReDoS) attacks exploit the fact that some regular expression implementations may reach extreme situations that cause them to work very slowly, causing programs to hang for extended periods of time. If the regex itself is affected by a user input, the attacker can inject an evil regex themselves and make the system vulnerable. For the issues presented by sonarqube, the user did not have the ability to input regex, so the second possibility was dismissed. Instead, the regex was checked using a tool (https://github.com/NicolaasWeideman/RegexStaticAnalysis) sourced from the internet that is designed to detect exponential degree of ambiguity (EDA) and infinite degree of ambiguity (IDA) by inspecting the underlie NFA of a regular expression. These checks determine if the regex is vulnerables to exponential backtracking or polynomial backtracking, respectively. In the case of the regex that was brought to light by sonarqube, the tool detected that it did not contain EDA or IDA, so the vulnerability was dismissed in the end.

Generated Reports

Below are some examples of the generated reports and images of the progress as the reports were analyzed and issues were updated.


sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Search for projects and files...
BA
Neptune Bank App
master
July 20, 2021, 8:12 PM Version 1.0.0-SNAPSHOT
Overview
Issues
Security Reports
Measures
Code
Activity
Administration
My Issues
All
Bulk Change
to select issues
to navigate
1 / 8 issues
50min effort
Filters
Clear All Filters
Type
Vulnerability
Reset
Bug 9
Vulnerability 8
Code Smell 487
Security Hotspot 89
⌘ + click to add to selection
Severity
Blocker 0
Minor 7
Critical 1
Info 0
Major 0
Resolution
Status
Creation Date
Language
Rule
Standard
Tag
Directory
File
Assignee
Author
src/.../com/neptunebank/app/domain/Branch.java
Return a copy of "accounts".
Vulnerability Minor Open Not assigned 5min effort Comment
last year L111
cert, cwe, unpredictable
Store a copy of "accounts".
Vulnerability Minor Open Not assigned 5min effort Comment
last year L115
cert, cwe, unpredictable
Store a copy of "accounts".
Vulnerability Minor Open Not assigned 5min effort Comment
last year L132
cert, cwe, unpredictable
src/.../com/neptunebank/app/domain/User.java
Return a copy of "authorities".
Vulnerability Minor Open Not assigned 5min effort Comment
last year L193
cert, cwe, unpredictable
Store a copy of "authorities".
Vulnerability Minor Open Not assigned 5min effort Comment
last year L197
cert, cwe, unpredictable
src/.../com/neptunebank/app/service/FileUploadService.java
Annotate this member with "@Autowired", "@Resource", "@Inject", or "@Value", or remove it.
Vulnerability Critical Open Not assigned 15min effort Comment
last year L16
owasp-a3, spring
src/.../neptunebank/app/service/dto/UserDTO.java
Return a copy of "authorities".
Vulnerability Minor Open Not assigned 5min effort Comment
last year L174
cert, cwe, unpredictable
Store a copy of "authorities".
Vulnerability Minor Open Not assigned 5min effort Comment
last year L178
cert, cwe, unpredictable
8 of 8 shown
SonarQube™ technology is powered by SonarSource SA
Community Edition - Version 7.7 (build 23042) - LGPL v3 - Community - Documentation - Get Support - Plugins - Web API - About


Security Hotspot
Projects
Issues
Rules
Quality Profiles
Quality Gates
Search for projects and files...
BA
Neptune Bank App
master
July 20, 2021, 8:12 PM Version 1.0.0-SNAPSHOT
Overview
Issues
Security Reports
Measures
Code
Activity
Administration
My Issues
All
Bulk Change
to select issues
to navigate
1 / 89 issues
1d 6h effort
Filters
Clear All Filters
Type
Security Hotspot
Reset
Bug 9
Vulnerability 8
Code Smell 487
Security Hotspot 89
⌘ + click to add to selection
Severity
Blocker 0
Minor 0
Critical 89
Info 0
Major 0
Resolution
Status
Creation Date
Language
Rule
Standard
Tag
Directory
File
Assignee
Author
src/main/java/com/neptunebank/app/NeptuneBankApp.java
Make sure that command line arguments are used safely here.
Security Hotspot Open Comment
last year L58
cwe, owasp-a1, sans-top25-insecure
src/.../com/neptunebank/app/config/SecurityConfiguration.java
Make sure disabling Spring Security's CSRF protection is safe here.
Security Hotspot Open Comment
last year L60
cwe, owasp-a6, sans-top25-insecure, ...
src/.../com/neptunebank/app/config/WebConfigurer.java
Make sure this file handling is safe here.
Security Hotspot Open Comment
last year L94
cert, cwe, owasp-a1, owasp-a3, sans-...
Make sure this file handling is safe here.
Security Hotspot Open Comment
last year L112
cert, cwe, owasp-a1, owasp-a3, sans-...
src/.../com/neptunebank/app/domain/User.java
Make sure that using a regular expression is safe here.
Security Hotspot Open Comment
last year L36
cwe, owasp-a1, sans-top25-porous
src/.../com/neptunebank/app/security/DomainUserDetailsService.java
Make sure that Permissions are controlled safely here.
Security Hotspot Open Comment
last year L56
owasp-a5, sans-top25-porous
src/.../com/neptunebank/app/service/FileUploadService.java
Make sure this file handling is safe here.
Security Hotspot Open Comment
last year L18
cert, cwe, owasp-a1, owasp-a3, sans-...
Make sure this file handling is safe here.
Security Hotspot Open Comment
last year L21
cert, cwe, owasp-a1, owasp-a3, sans-...
Make sure this file handling is safe here.
Security Hotspot Open Comment
last year L25
cert, cwe, owasp-a1, owasp-a3, sans-...
eam-01&resolved=false&types=SECURITY_HOTSPOT#



Make sure this file handling is safe here. Manual — last year L35
Vulnerability Critical Open Ben Austin Comment — cert, cwe, owasp-a1, owasp-a3, sans-…
Ben Austin Need to address this entire page. No file validation. Users can upload anything. — 10 hours ago

Make sure this file handling is safe here. Manual — last year L41
Vulnerability Critical Open Ben Austin Comment — cert, cwe, owasp-a1, owasp-a3, sans-…
Ben Austin Need to address this entire page. No file validation. Users can upload anything. — 10 hours ago

src/.../neptunebank/app/service/dto/UserDTO.java

Make sure that using a regular expression is safe here. Manual — last year L23
Vulnerability Minor Open David Bishop Comment — cwe, owasp-a1, sans-top25-porous
Ben Austin Unsure, assigned to David — 10 hours ago

Return a copy of "authorities". — last year L174
Vulnerability Major Resolved (Fixed) Ben Austin 5min effort Comment — cert, cwe, unpredictable
Ben Austin Seems like a simple fix, will address. — yesterday
Ben Austin Fixed by checking for null parameters. — 11 seconds ago

Store a copy of "authorities". — last year L178
Vulnerability Major Resolved (Fixed) Ben Austin 5min effort Comment — cert, cwe, unpredictable
Ben Austin Seems like a simple fix, will address. — yesterday
Ben Austin Fixed by checking for null parameters. — now

10 of 10 shown

Make sure this file handling is safe here. — last year L26
Security Hotspot Resolved (Won't fix) Comment — cert, cwe, owasp-a1, owasp-a3, sans-…
Ben Austin Need to address this entire page. No file validation. Users can upload anything. — 14 hours ago
Ben Austin Non-issue. — 1 minute ago
Ben Austin Added validation to client and server sides. — 53 seconds ago

Make sure this file handling is safe here. — last year L30
Security Hotspot Resolved (Won't fix) Comment — cert, cwe, owasp-a1, owasp-a3, sans-…
Ben Austin Need to address this entire page. No file validation. Users can upload anything. — 14 hours ago
Ben Austin Added validation to client and server sides. — 53 seconds ago

Make sure this file handling is safe here. — last year L43
Security Hotspot Resolved (Won't fix) Comment — cert, cwe, owasp-a1, owasp-a3, sans-…
Ben Austin Need to address this entire page. No file validation. Users can upload anything. — 14 hours ago
Ben Austin Added validation to client and server sides. — 39 seconds ago

Make sure this file handling is safe here. — last year L52
Security Hotspot Resolved (Won't fix) Comment — cert, cwe, owasp-a1, owasp-a3, sans-…
Ben Austin Need to address this entire page. No file validation. Users can upload anything. — 14 hours ago
Ben Austin Added validation to client and server sides. — 37 seconds ago

Penetration Test

ZAP Logs and Screenshots

The active Scan was run twice. We did this as we encountered errors in setting up the spider. When setting up the spider the message “unauthorized” or “forbidden”. We had followed the procedure that was given in the lab and were unable to not receive unauthorized errors. We were not able to figure out if this was intended, however, to be safe we executed a manual scan as an Admin multiple times to ensure we mapped all paths. A summary of the initial ZAP Scanning Report can be shown below, with a large amount of alerts and vulnerabilities being tracked.

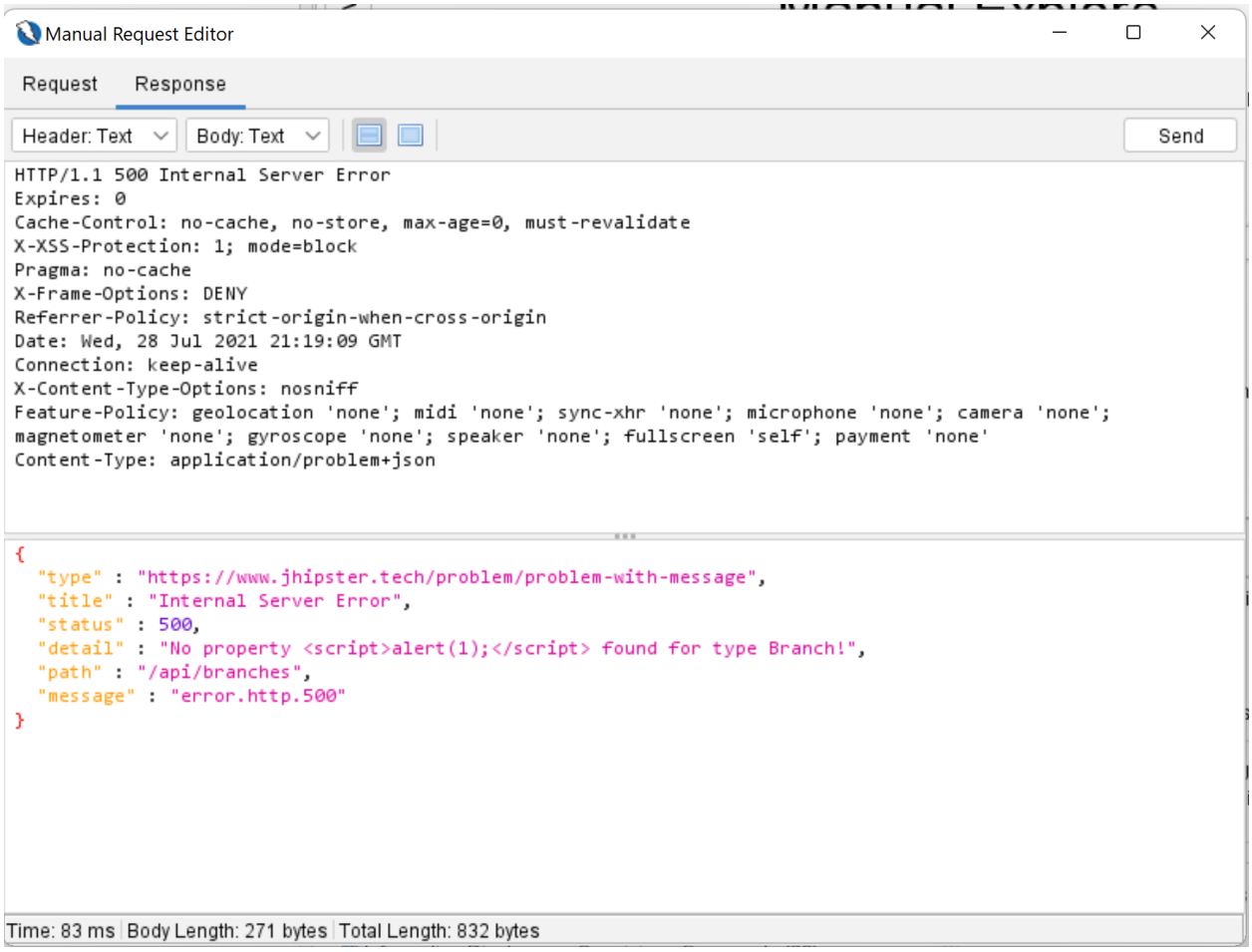
ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	2
Medium	2
Low	12
Informational	8

Alerts

Name	Risk Level	Number of Instances
Path Traversal	High	1
SQL Injection	High	1
Cross-Domain Misconfiguration	Medium	2
Cross Site Scripting Weakness (Reflected in JSON Response)	Low	6
Incomplete or No Cache-control Header Set	Low	40
X-Content-Type-Options Header Missing	Low	31
Charset Mismatch	Informational	1
Information Disclosure - Suspicious Comments	Informational	21
Timestamp Disclosure - Unix	Informational	210



The first scan was performed directly after a manual scan and the table produced was copied below.

	Strength	Progress	Elapsed	Reqs	Alerts	Status
Analyser			00:01.765	12		
Plugin						
Path Traversal	High	100	02:43.200	301	0	Completed
Remote File Inclusion	High	100	02:01.776	200	0	Completed
Source Code Disclosure - /WEB-INF folder	High	100	00:00.176	2	0	Completed
External Redirect	High	100	00:37.584	110	0	Completed

Server Side Include	High	100	00:07.775	40	0	Completed
Cross Site Scripting (Reflected)	High	100	00:07.350	38	1	Completed
Cross Site Scripting (Persistent) - Prime	High	100	00:06.633	10	0	Completed
Cross Site Scripting (Persistent) - Spider	High	100	00:09.275	22	0	Completed
Cross Site Scripting (Persistent)	High	100	00:03.920	0	0	Completed
SQL Injection	High	100	00:38.204	433	2	Completed
Server Side Code Injection	High	100	00:06.835	50	0	Completed
Remote OS Command Injection	High	100	01:08.450	510	0	Completed
Directory Browsing	High	100	00:05.350	22	0	Completed
Buffer Overflow	High	100	00:00.000	0	0	Skipped, scanner does not target selected technologies .
Format String Error	High	100	00:00.000	0	0	Skipped, scanner does not target selected technologies .
CRLF Injection	High	100	00:11.843	70	0	Completed
Parameter Tampering	High	100	00:06.931	46	0	Completed

ELMAH Information Leak	High	100	00:00.073	1	0	Completed
.htaccess Information Leak	High	100	00:02.989	7	0	Completed
Script Active Scan Rules	High	100	00:00.015	0	0	Skipped, no scripts enabled.
Cross Site Scripting (DOM Based)	High	100	14:06.838	917	0	Completed
SOAP Action Spoofing	High	100	00:00.377	0	0	Completed
SOAP XML Injection	High	100	00:02.229	0	0	Completed
Totals			**22:31.649**	**2917**	**3**	

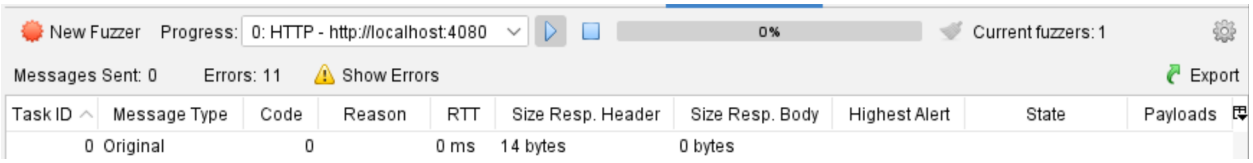
The second scan was performed after a manual scan and a forced browse directory attack. The table produced was copied below. This attack took so much longer as the forced browse attack created so many new directories.

	Strength	Progress	Elapsed	Reqs	Alerts	Status
Analyser			13:41.073	44757		
Plugin						
Path Traversal	Medium	100	86:36.477	844	1	Completed
Remote File Inclusion	Medium	100	91:00.060	530	0	Completed
Source Code Disclosure - /WEB-INF folder	Medium	100	00:00.049	2	0	Completed
External Redirect	Medium	100	80:08.076	477	0	Completed
Server Side Include	Medium	100	79:49.982	212	0	Completed

Cross Site Scripting (Reflected)	Medium	100	79:09.687	182	5	Completed
Cross Site Scripting (Persistent) - Prime	Medium	100	79:36.839	53	0	Completed
Cross Site Scripting (Persistent) - Spider	Medium	100	34:26.113	44795	0	Completed
Cross Site Scripting (Persistent)	Medium	100	78:04.686	0	0	Completed
SQL Injection	Medium	100	167:44.209	1363	0	Completed
Server Side Code Injection	Medium	100	82:01.189	265	0	Completed
Remote OS Command Injection	Medium	100	253:40.064	1632	0	Completed
Directory Browsing	Medium	100	70:43.284	44795	0	Completed
Buffer Overflow	Medium	100	00:00.000	0	0	Skipped, scanner does not target selected technologies .
Format String Error	Medium	100	00:00.000	0	0	Skipped, scanner does not target selected technologies .
CRLF Injection	Medium	100	78:26.755	371	0	Completed
Parameter Tampering	Medium	100	78:51.028	329	0	Completed
ELMAH Information Leak	Medium	100	00:00.031	1	0	Completed

.htaccess Information Leak	Medium	100	09:43.565	22383	0	Completed
Script Active Scan Rules	Medium	100	00:00.000	0	0	Skipped, no scripts enabled.
Cross Site Scripting (DOM Based)	Medium	100	58:10.617	4027	0	Skipped, by user action.
SOAP Action Spoofing	Medium	100	07:19.707	0	0	Completed
SOAP XML Injection	Medium	100	80:23.887	0	0	Completed
Totals			**1509:39.640**	**538008**	**6**	

Zap was also used to run fuzz and a screenshot of that running below is shown. Both the already available file fuzzer disctories were used to fuzz the input fields.



Application Scan for Vulnerabilities

The following table analyzes the logs captured from the ZAP penetration testing. Focusing on the more uncommon alerts, each alert was identified to be a valid alert or an invalid alert; a false positive. This is displayed by the “Validity” column in the table below. Most alerts were identified to not be in need of developer attention, with only a few being true positives.

Alert Name	ZAP Risk Level	Actual Risk Level	Validity
Path Traversal	High(low)	high(low)	Valid
SQL Injection	High(medium)	high(high)	Valid
Cross-Domain Misconfigurations	Medium(medium)	low(low)	Invalid
Cross Site Scripting Weakness (Reflected in JSON response)	Low(low)	high(high)	Valid

Incomplete or No Cache-Control Header Sets	Low(medium)	low(low)	Invalid
X-Content-Type-Options Headers Missing	Low(medium)	low(low)	Invalid
Charset Mismatch	Informational(Low)	low(low)	Invalid
Information Disclosure - Suspicious Comments	Informational(Low)	Informational(high)	Valid
Timestamp Disclosures - Unix	Informational(Low)	Low	Invalid

Severity Classification of Vulnerabilities Found

In the above table, each alert was analyzed on the basis of its assigned risk level, given by ZAP, and allocated a new column, “Actual Risk Level”, after manual verification.The ZAP-assigned and manually-assigned severity of the alerts involved high, medium, low and informational levels. For the most part, the ZAP assigned severity was accurate and did not need to be updated after manual analysis. In terms of classification, high severity vulnerabilities are those that need to be dealt with immediately as they pose a high risk to threats from potential attackers. Likewise, medium severity vulnerabilities are those that should be dealt with as soon as possible. Most of the false positive alerts were assigned a low severity vulnerability after verification. This means that there is little to no priority for fixing these vulnerabilities as they do not pose any critical threats to the application.

High severity Vulnerabilities

Out of all of the vulnerabilities identified by ZAP, 4 were found to be true-positive, high severity vulnerabilities that should be attended to as soon as possible. These vulnerabilities involved Path Traversal, SQL Injection, XSS, and Suspicious Comments. The Path Traversal vulnerability of the “/api/account” URL is high severity as attackers could be able to access the repository structure of the application, including everything contained within the repository, such as confidential files. Secondly, there is an SQL Injection Vulnerability of the “/api/authenticate” endpoint in the “password” parameter. Attackers would easily be able to look up different kinds of SQL Injection strings and input them into the correct form field to cause havoc within the database. Furthermore, there is a XSS vulnerability on the “/api/branches” URL where an attacker would be able to append a script to the URL. Lastly, there is a Suspicious Comments vulnerability in a lot of 3rd party libraries that are being imported into the application. Since some of the frontend’s source code can be seen through the “inspect” webtool, an attacker that sees these comments may be able to understand weaknesses of the system more thoroughly.

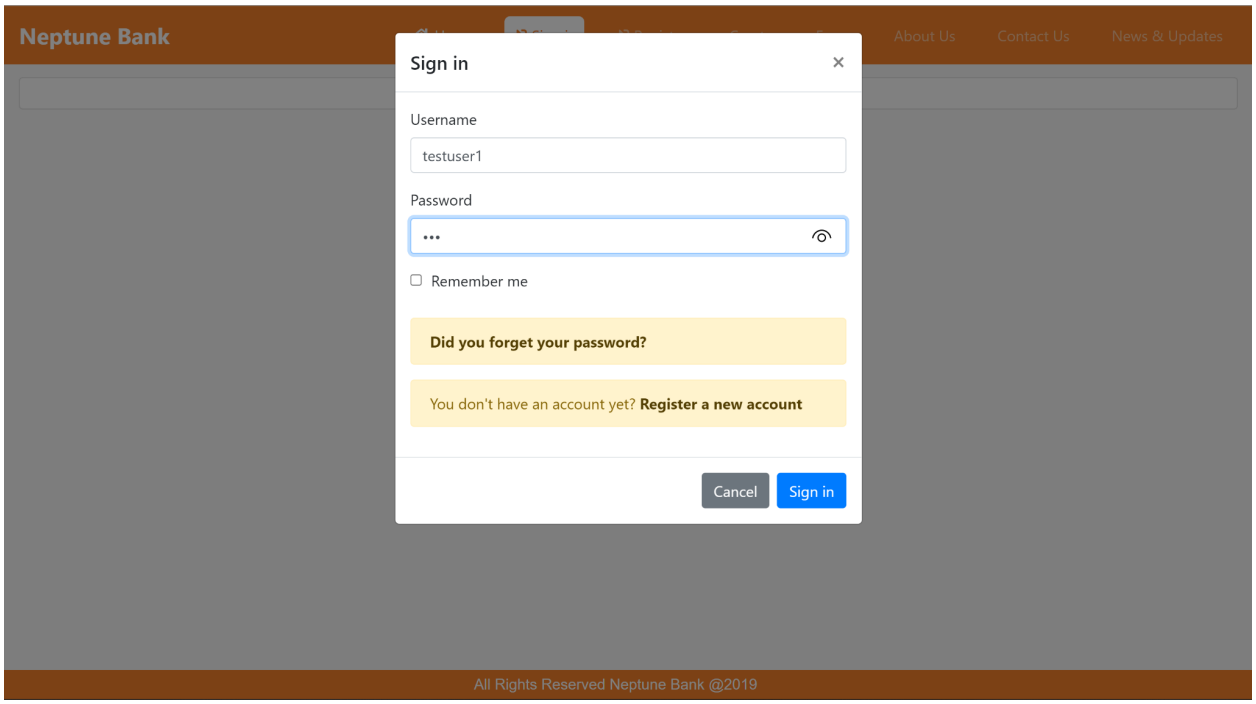
Semi Manual Attacks

A few semi manual attacks were conducted:

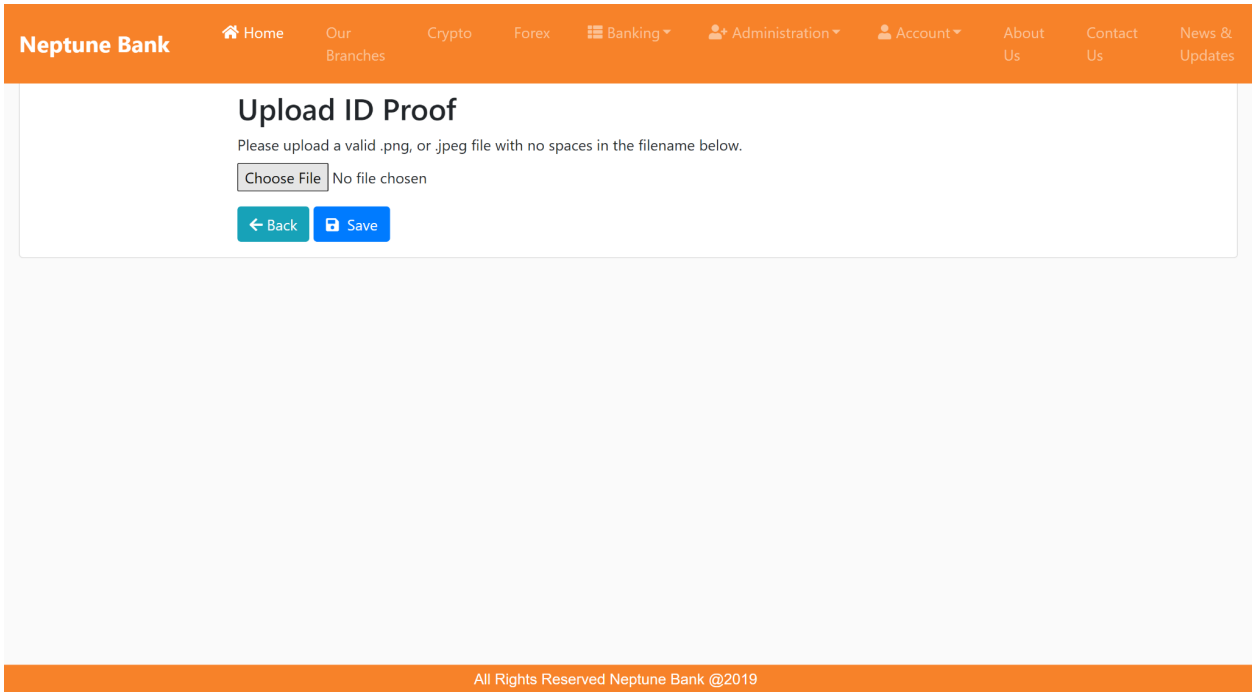
A SQL injection of the login page was conducted. The following injections were attempted and were successful similar to what is shown in the figure below. Since we had deleted the user with an Admin name we used one of the users we knew existed.

or 1=1--
admin' --

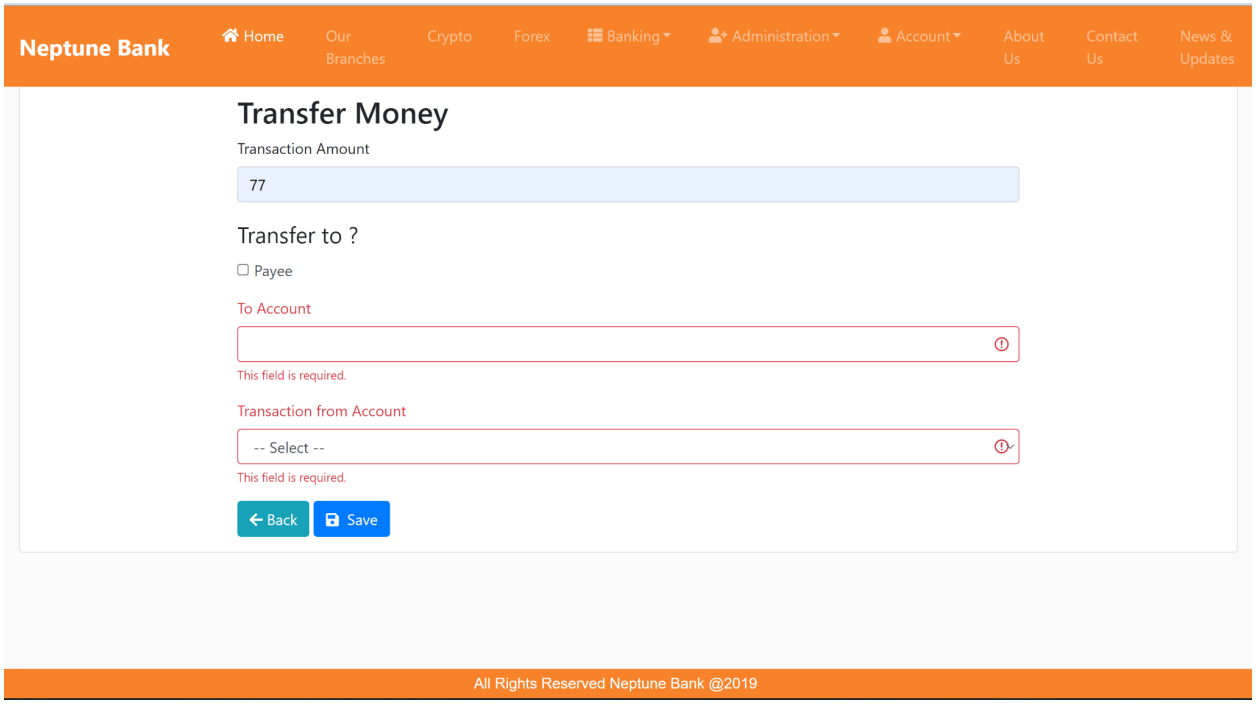
These injections allowed us to bypass the login screen and become authenticated. This was because the strings accepted weren't escaped.



The second attack executed was a stored XSS attack. Since the Id proof uploading accepts all files, script files could be uploaded that executes using an already authenticated account. A MYSQL script was injected, if we assume that the script wasn't noticed it can be very malicious. In our case mvnw clean had to be run as we purged the database. We used the DROP DATABASE command in our script. We also just assumed the attacker came about the sql table through incorrect error handling.



When transferring money another sql injection attack is possible. The to account can be manipulated and be invalid but also allowed to execute. The attacker in this case can add an entire statement or drop the database similar to the above example. This is because the attacker is able to use the --and ; to end the expected statement and enter a new sql statement following this.

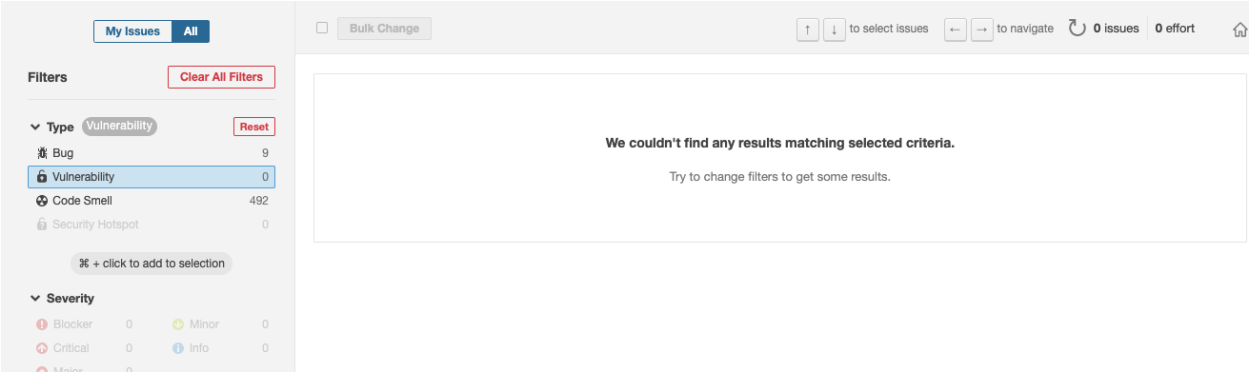


Bug Fixes

All bug fix information for Sonarqube scans is contained within Sonarqube, and an explanation of the fixes and process are outlined in the “Static Source Code Analysis” section above.

Application Rescanned After Bug Fixes

After completing bug fixes and assessing threats on Sonarqube, the resulting scan dashboard can be seen, and 0 Vulnerabilities and 0 Security Hotspot reports.



The image below shows the ZAP Scanning summary after resolving a few of the vulnerabilities within neptunebank. The SQL Injection was resolved through escaping all data that was sent, the XSS issues were resolved through importing dompurify, a 3rd party package that encodes URLs to prevent XSS attacks from taking place. Unfortunately, a fix for the Suspicious Comments could not be developed, as the application creates javascript files that hold these comments, but we do not have access to these files and simply removing them from being created poses larger risks to our application. The remaining Alerts listed below are all out of scope; they all exist within the 3rd party applications and libraries linked to our application, and therefore we could not resolve them. Furthermore, the Path Traversal Vulnerability as exposed within the initial ZAP Scan was found to be a false positive.

ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	2
Low	11
Informational	6

Alerts

Name	Risk Level	Number of Instances
Cross-Domain Misconfiguration	Medium	2
Incomplete or No Cache-control and Pragma HTTP Header Set	Low	14
Private IP Disclosure	Low	1
X-Content-Type-Options Header Missing	Low	29
Information Disclosure - Suspicious Comments	Informational	4
Timestamp Disclosure - Unix	Informational	107

Resolution

Comparison of SonarQube and ZAP (Zed Attack Proxy)

While both technologies are very useful for security testing they have their respective props and cons and specific use cases which might sway a developer. A good example to illustrate why a security tester might choose to use SonarCube, for example, is that it has the ability to integrate directly with CI tools like Jenkins which has been used previously in this course. Furthermore SonarQube has the capability to analyze pull requests, integration to this scale is lacking in the ZAP technology. That being said, ZAP is developed by the Open Web Application Security Project (OWASP) [1] and has more functionality in the way of possible security tests that it can run, exceeding that of SonarQube. For example, ZAP can detect missing security headers and anti-cross site scripting forgery tokens in requests made. In our experience as a group ZAP was also more intuitive and easier to use when scanning and detecting vulnerabilities in the repository.

How can Testing be Improved?

Firstly, testing our application can be greatly improved by having a more mature approach to testing the different user roles When dealing with multiple user roles the SQL gets more complex and there is a greater possibility for SQL injection and unauthorized users gaining elevated privileges. But quite possibly the best thing we can do to improve testing is to do it earlier in the development process. By leaving security testing to the end there is a great chance that a good portion of the application will need to be redeveloped to fix glaring flaws that could have been caught earlier on. By testing early and by building security, we are able to make sure the application is secure at every level and there is a much smaller chance that any attackers could gain anything of value from trying to penetrate the application.

Bibliography

[1]. “Top 10 Open Source Security Testing Tools for Web Applications (Updated),” Hackr.io. [Online]. Available: https://hackr.io/blog/top-10-open-source-security-testing-tools-for-web-applications. [Accessed: 31-Jul-2021].

[2] "OWASP ZAP", Zaproxy.org, 2021. [Online]. Available: https://www.zaproxy.org/docs/desktop/start/features/authentication/. [Accessed: 31- Jul- 2021]