Final Project

CS371 Software Security

Habib University

Spring 2024

Instructor: Muhammad Nadeem

Team Members: Abeeha Zehra, Laiba Zehra

Table of Contents

1.SonarQube:

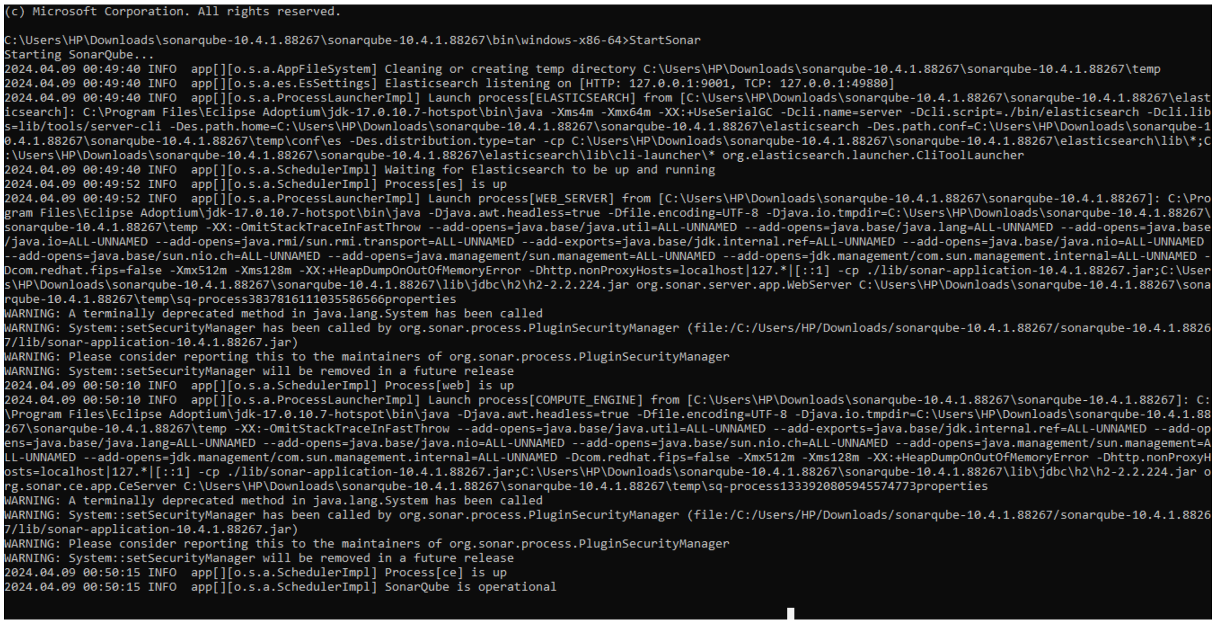
For this project, we were required to analyze the source code of the selected project, analyze the vulnerabilities in them, and provide mitigations for them. Hence, for the analysis of the project we chose SonarQube which is a code quality assurance tool that scans and analyzes the code and provides a detailed report on it. This tool can perform both static as well as dynamic analysis of our code. It analyzes the code in such detail that it can easily identify where things went wrong in the code, and what issues are present in the code, from styling issues to excessively complex code.

Hence, for our project we downloaded the community version of this tool from here. The version of the tool that we worked on is 10.4.1 and provides features such as:

- Static code analysis for 19 languages including Java, C#, JavaScript, Python, CSS, PHP, GO, etc.
- Detects bugs and vulnerabilities.
- Reviews security hotspots in codebase.
- It provides various code quality metrics to measure the health and maintainability of the codebase over time.
- It also identifies and tracks code smells, which basically means it can help us indicate potential issues or areas where we can improve.

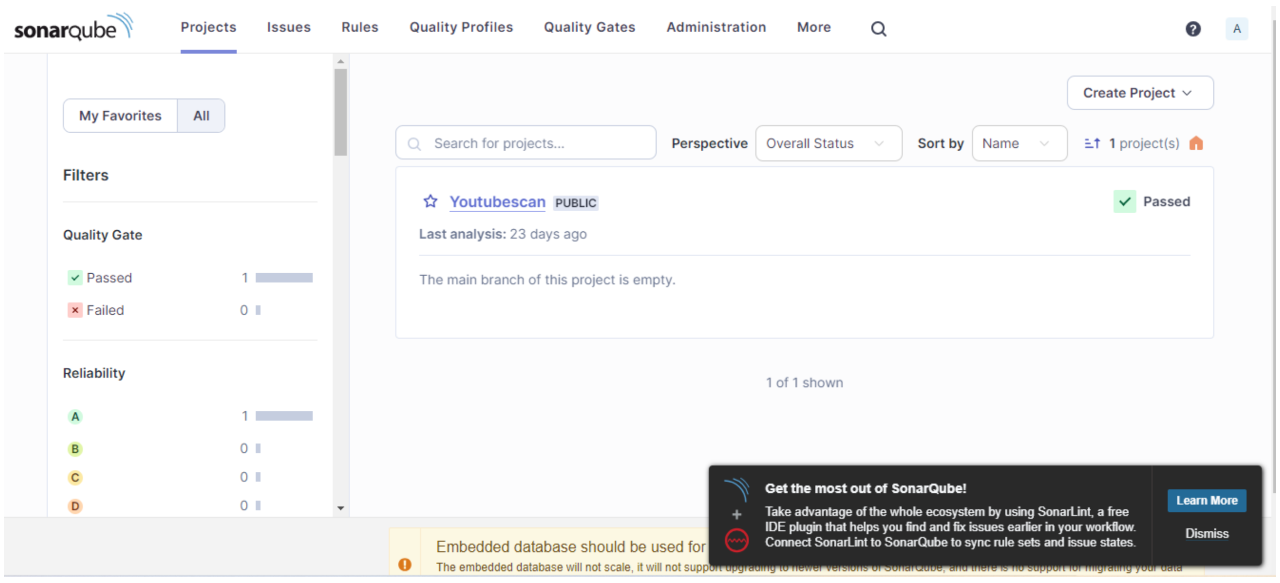
Therefore, by using SonarQube for this project, we will enhance quality, security, and analyze the area of development in the code base.

We set up this tool locally on our machines and accessed it locally as shown in Figure 1 and 2.



```
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP\Downloads\sonarqube-10.4.1.88267\sonarqube-10.4.1.88267\bin\windows-x86-64>StartSonar
Starting SonarQube...
2024.04.09 00:49:40 INFO  app[][o.s.a.AppFileSystem] Cleaning or creating temp directory C:\Users\HP\Downloads\sonarqube-10.4.1.88267\sonarqube-10.4.1.88267\temp
2024.04.09 00:49:40 INFO  app[][o.s.a.es.EsSettings] Elasticsearch listening on [HTTP: 127.0.0.1:9001, TCP: 127.0.0.1:49880]
2024.04.09 00:49:40 INFO  app[][o.s.a.ProcessLauncherImpl] Launch process[ELASTICSEARCH] from [C:\Users\HP\Downloads\sonarqube-10.4.1.88267\sonarqube-10.4.1.88267\elast
icsearch]: C:\Program Files\Eclipse Adoptium\jdk-17.0.10.7-hotspot\bin\java -Xms4m -Xmx64m -XX:+UseSerialGC -Dcli.name=server -Dcli.script=./bin/elasticsearch -Dcli.lib
s=lib/tools/server-cli -Des.path.home=C:\Users\HP\Downloads\sonarqube-10.4.1.88267\sonarqube-10.4.1.88267\elasticsearch -Des.path.conf=C:\Users\HP\Downloads\sonarqube-1
0.4.1.88267\sonarqube-10.4.1.88267\temp\conf\es -Des.distribution.type=tar -cp C:\Users\HP\Downloads\sonarqube-10.4.1.88267\sonarqube-10.4.1.88267\elasticsearch\lib\*;C
:\Users\HP\Downloads\sonarqube-10.4.1.88267\sonarqube-10.4.1.88267\elasticsearch\lib\cli-launcher\* org.elasticsearch.launcher.CliToolLauncher
2024.04.09 00:49:40 INFO  app[][o.s.a.SchedulerImpl] Waiting for Elasticsearch to be up and running
2024.04.09 00:49:52 INFO  app[][o.s.a.SchedulerImpl] Process[es] is up
2024.04.09 00:49:52 INFO  app[][o.s.a.ProcessLauncherImpl] Launch process[WEB_SERVER] from [C:\Users\HP\Downloads\sonarqube-10.4.1.88267\sonarqube-10.4.1.88267]: C:\Pro
gram Files\Eclipse Adoptium\jdk-17.0.10.7-hotspot\bin\java -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Djava.io.tmpdir=C:\Users\HP\Downloads\sonarqube-10.4.1.88267\
sonarqube-10.4.1.88267\temp -XX:-OmitStackTraceInFastThrow --add-opens=java.base/java.util=ALL-UNNAMED --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base
/java.io=ALL-UNNAMED --add-opens=java.rmi/sun.rmi.transport=ALL-UNNAMED --add-exports=java.base/jdk.internal.ref=ALL-UNNAMED --add-opens=java.base/java.nio=ALL-UNNAMED
--add-opens=java.base/sun.nio.ch=ALL-UNNAMED --add-opens=java.management/sun.management=ALL-UNNAMED --add-opens=jdk.management/com.sun.management.internal=ALL-UNNAMED -
Dcom.redhat.fips=false -Xmx512m -Xms128m -XX:+HeapDumpOnOutOfMemoryError -Dhttp.nonProxyHosts=localhost|127.*|[::1] -cp ./lib/sonar-application-10.4.1.88267.jar;C:\User
s\HP\Downloads\sonarqube-10.4.1.88267\sonarqube-10.4.1.88267\lib\jdbc\h2\h2-2.2.224.jar org.sonar.server.app.WebServer C:\Users\HP\Downloads\sonarqube-10.4.1.88267\sona
rqube-10.4.1.88267\temp\sq-process3837816111035586566properties
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by org.sonar.process.PluginSecurityManager (file:/C:/Users/HP/Downloads/sonarqube-10.4.1.88267/sonarqube-10.4.1.8826
7/lib/sonar-application-10.4.1.88267.jar)
WARNING: Please consider reporting this to the maintainers of org.sonar.process.PluginSecurityManager
WARNING: System::setSecurityManager will be removed in a future release
2024.04.09 00:50:10 INFO  app[][o.s.a.SchedulerImpl] Process[web] is up
2024.04.09 00:50:10 INFO  app[][o.s.a.ProcessLauncherImpl] Launch process[COMPUTE_ENGINE] from [C:\Users\HP\Downloads\sonarqube-10.4.1.88267\sonarqube-10.4.1.88267]: C:
\Program Files\Eclipse Adoptium\jdk-17.0.10.7-hotspot\bin\java -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Djava.io.tmpdir=C:\Users\HP\Downloads\sonarqube-10.4.1.88
267\sonarqube-10.4.1.88267\temp -XX:-OmitStackTraceInFastThrow --add-opens=java.base/java.util=ALL-UNNAMED --add-exports=java.base/jdk.internal.ref=ALL-UNNAMED --add-op
ens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.nio=ALL-UNNAMED --add-opens=java.base/sun.nio.ch=ALL-UNNAMED --add-opens=java.management/sun.management=A
LL-UNNAMED --add-opens=jdk.management/com.sun.management.internal=ALL-UNNAMED -Dcom.redhat.fips=false -Xmx512m -Xms128m -XX:+HeapDumpOnOutOfMemoryError -Dhttp.nonProxyH
osts=localhost|127.*|[::1] -cp ./lib/sonar-application-10.4.1.88267.jar;C:\Users\HP\Downloads\sonarqube-10.4.1.88267\sonarqube-10.4.1.88267\lib\jdbc\h2\h2-2.2.224.jar o
rg.sonar.ce.app.CeServer C:\Users\HP\Downloads\sonarqube-10.4.1.88267\sonarqube-10.4.1.88267\temp\sq-process1333920805945574773properties
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by org.sonar.process.PluginSecurityManager (file:/C:/Users/HP/Downloads/sonarqube-10.4.1.88267/sonarqube-10.4.1.8826
7/lib/sonar-application-10.4.1.88267.jar)
WARNING: Please consider reporting this to the maintainers of org.sonar.process.PluginSecurityManager
WARNING: System::setSecurityManager will be removed in a future release
2024.04.09 00:50:15 INFO  app[][o.s.a.SchedulerImpl] Process[ce] is up
2024.04.09 00:50:15 INFO  app[][o.s.a.SchedulerImpl] SonarQube is operational
```

Figure 1: Locally setting up the server



Last analysis: 23 days ago

The main branch of this project is empty.

Get the most out of SonarQube!

Take advantage of the whole ecosystem by using SonarLint, a free IDE plugin that helps you find and fix issues earlier in your workflow. Connect SonarLint to SonarQube to sync rule sets and issue states.

Embedded database should be used for

The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data

Figure 2: Accessing the tool locally

2.Software introduction:

Source Code URL: https://github.com/monero-project/monero
Monero version: v0.18.1.2

2.1 Monero:

Monero is a cryptocurrency designed to offer users privacy through unparalleled anonymity, security and decentralization. Launched in 2014, Monero (XMR) has quickly gained traction as a leading digital currency, thanks to its innovative approach to blockchain technology. It ensures that every transaction protects the purchases, receipts and transfers of its users. It uses many tools to ensure this kind of privacy and transactions are validated through a proof-of-work algorithm. Through a combination of ring signatures, stealth addresses, and optional ambiguity measures, Monero ensures that transactions remain untraceable and private by default.
While it is a great tool for us, there have been problems such as crypto jacking and its illicit use. However, it remains a powerful tool for users who value financial sovereignty and anonymity.

The creators of Monero have kept it open source so that other developers can make changes to it and then those changes are accepted or rejected by the main developers of the tool. While this tool ensures privacy, it may have weaknesses and vulnerabilities that can be a result of any implementation done by an individual or a group. For such a tool that has a lot of sensitive information about users, all vulnerabilities need to be mitigated. Hence, that is what we will be doing in this project to give relevant information about the vulnerabilities we may find through static analysis on SonarQube.

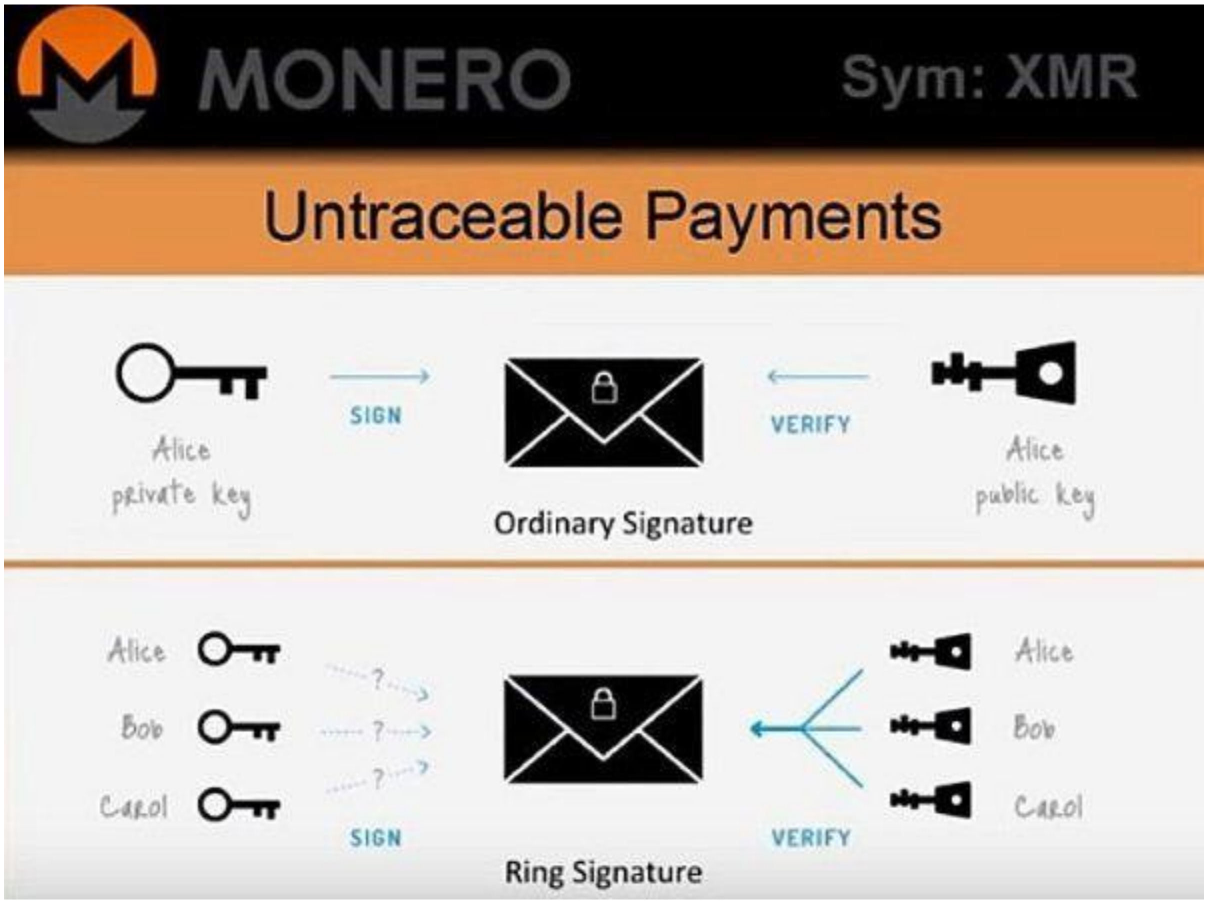


Figure 3: Working of Monero

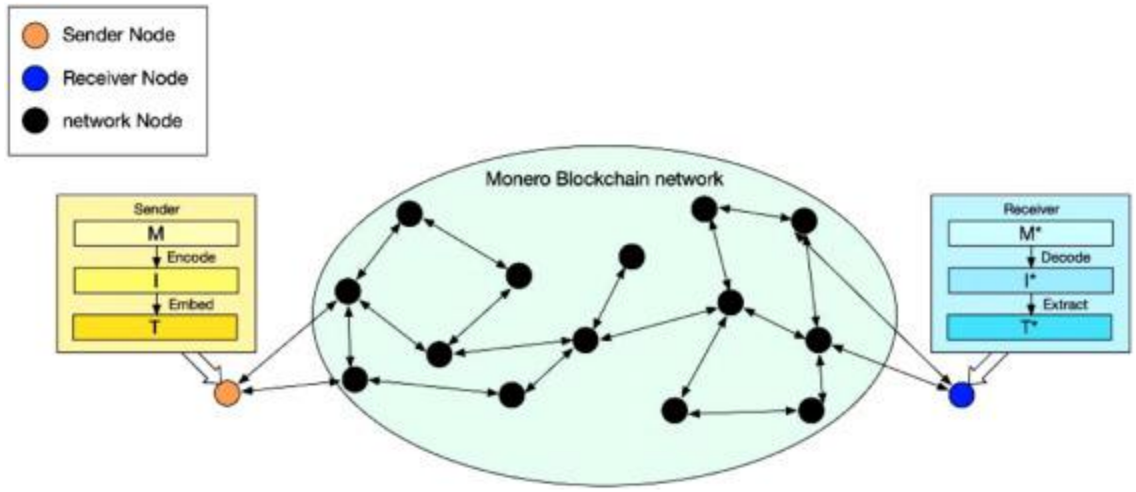
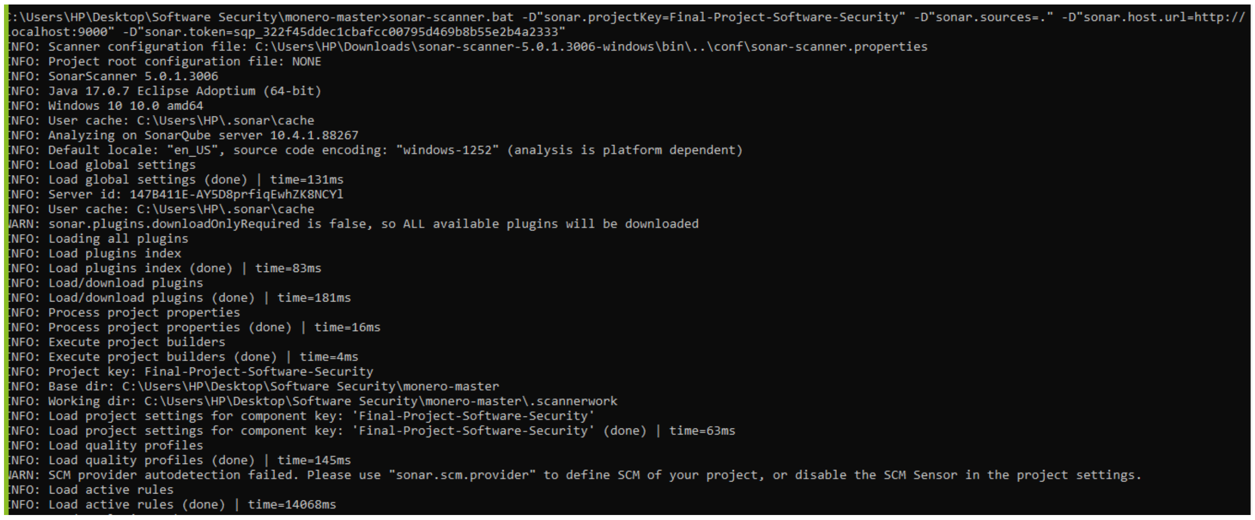


Figure 4: Flow diagram of Monero

3. Static Analysis on SonarQube:

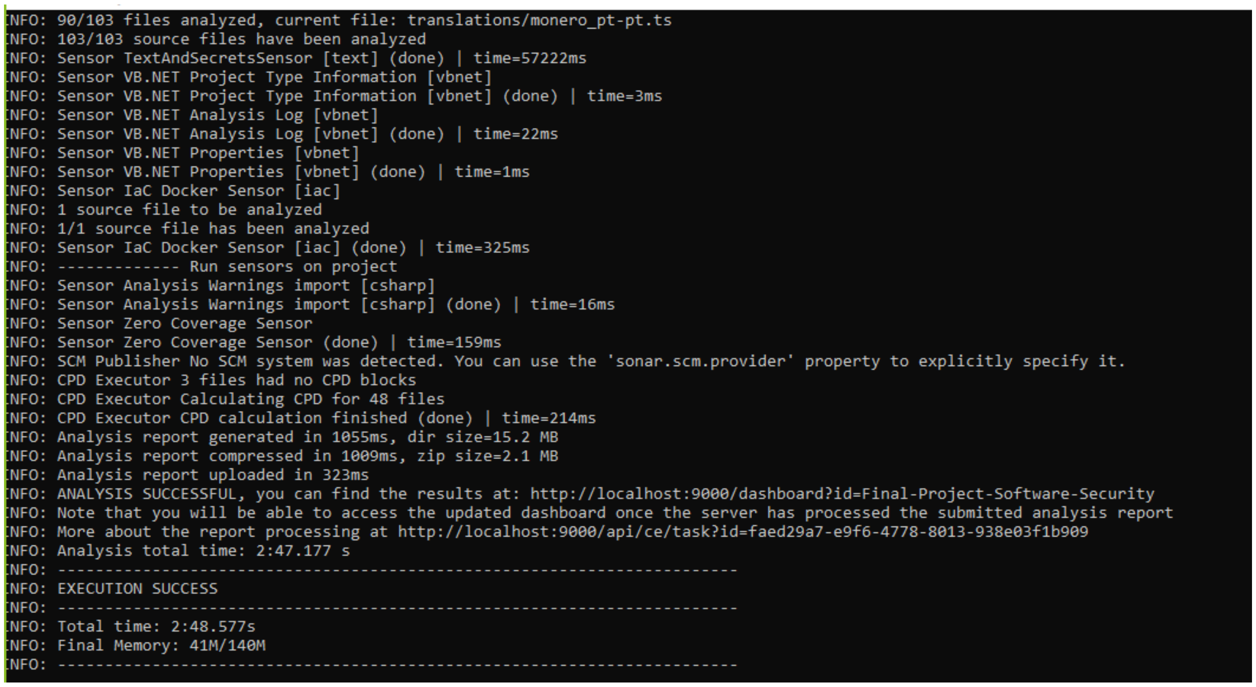
After setting up the SonarQube locally on our PC and downloading the source code of Monero, we performed the analysis on it as shown in Figure 5.



```
:\Users\HP\Desktop\Software Security\monero-master>sonar-scanner.bat -D"sonar.projectKey=Final-Project-Software-Security" -D"sonar.sources=." -D"sonar.host.url=http://
ocalhost:9000" -D"sonar.token=sqp_322f45ddec1cbafcc00795d469b8b55e2b4a2333"
NFO: Scanner configuration file: C:\Users\HP\Downloads\sonar-scanner-5.0.1.3006-windows\bin\..\conf\sonar-scanner.properties
NFO: Project root configuration file: NONE
NFO: SonarScanner 5.0.1.3006
NFO: Java 17.0.7 Eclipse Adoptium (64-bit)
NFO: Windows 10 10.0 amd64
NFO: User cache: C:\Users\HP\.sonar\cache
NFO: Analyzing on SonarQube server 10.4.1.88267
NFO: Default locale: "en_US", source code encoding: "windows-1252" (analysis is platform dependent)
NFO: Load global settings
NFO: Load global settings (done) | time=131ms
NFO: Server id: 147B411E-AY5D8prfiqEwhZK8NCYl
NFO: User cache: C:\Users\HP\.sonar\cache
ARN: sonar.plugins.downloadOnlyRequired is false, so ALL available plugins will be downloaded
NFO: Loading all plugins
NFO: Load plugins index
NFO: Load plugins index (done) | time=83ms
NFO: Load/download plugins
NFO: Load/download plugins (done) | time=181ms
NFO: Process project properties
NFO: Process project properties (done) | time=16ms
NFO: Execute project builders
NFO: Execute project builders (done) | time=4ms
NFO: Project key: Final-Project-Software-Security
NFO: Base dir: C:\Users\HP\Desktop\Software Security\monero-master
NFO: Working dir: C:\Users\HP\Desktop\Software Security\monero-master\.scannerwork
NFO: Load project settings for component key: 'Final-Project-Software-Security'
NFO: Load project settings for component key: 'Final-Project-Software-Security' (done) | time=63ms
NFO: Load quality profiles
NFO: Load quality profiles (done) | time=145ms
ARN: SCM provider autodetection failed. Please use "sonar.scm.provider" to define SCM of your project, or disable the SCM Sensor in the project settings.
NFO: Load active rules
NFO: Load active rules (done) | time=14068ms
```

Figure 5: Running the command to start the analysis

The total time to perform the analysis was 2:48.577s and the total memory consumed for the analysis was 41M as shown in Figure 6.

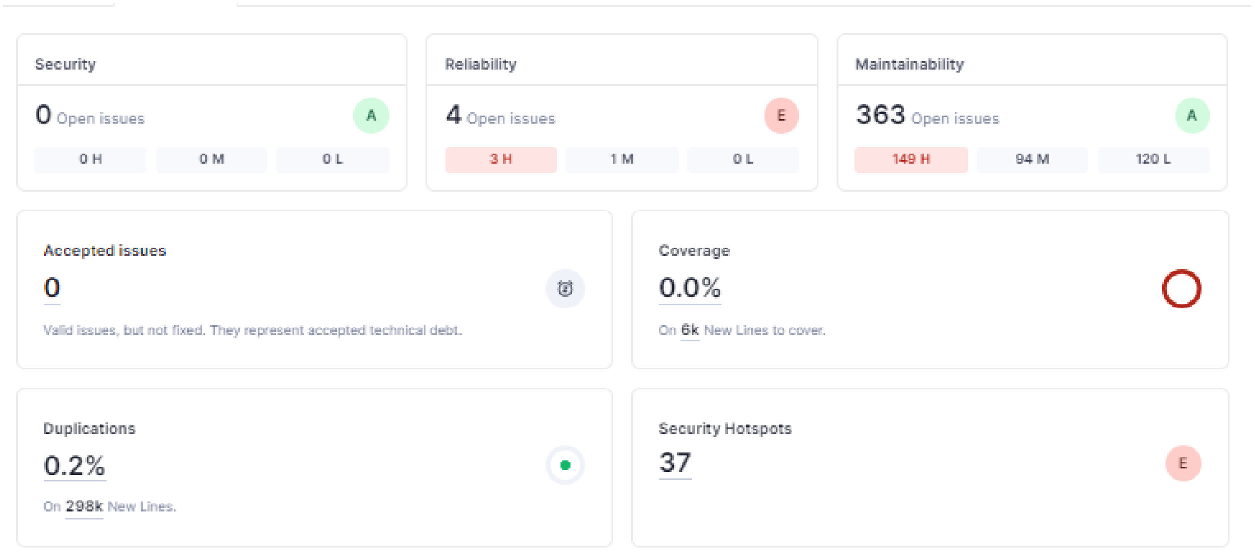


```
NFO: 90/103 files analyzed, current file: translations/monero_pt-pt.ts
NFO: 103/103 source files have been analyzed
NFO: Sensor TextAndSecretsSensor [text] (done) | time=57222ms
NFO: Sensor VB.NET Project Type Information [vbnet]
NFO: Sensor VB.NET Project Type Information [vbnet] (done) | time=3ms
NFO: Sensor VB.NET Analysis Log [vbnet]
NFO: Sensor VB.NET Analysis Log [vbnet] (done) | time=22ms
NFO: Sensor VB.NET Properties [vbnet]
NFO: Sensor VB.NET Properties [vbnet] (done) | time=1ms
NFO: Sensor IaC Docker Sensor [iac]
NFO: 1 source file to be analyzed
NFO: 1/1 source file has been analyzed
NFO: Sensor IaC Docker Sensor [iac] (done) | time=325ms
NFO: ------------- Run sensors on project
NFO: Sensor Analysis Warnings import [csharp]
NFO: Sensor Analysis Warnings import [csharp] (done) | time=16ms
NFO: Sensor Zero Coverage Sensor
NFO: Sensor Zero Coverage Sensor (done) | time=159ms
NFO: SCM Publisher No SCM system was detected. You can use the 'sonar.scm.provider' property to explicitly specify it.
NFO: CPD Executor 3 files had no CPD blocks
NFO: CPD Executor Calculating CPD for 48 files
NFO: CPD Executor CPD calculation finished (done) | time=214ms
NFO: Analysis report generated in 1055ms, dir size=15.2 MB
NFO: Analysis report compressed in 1009ms, zip size=2.1 MB
NFO: Analysis report uploaded in 323ms
NFO: ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=Final-Project-Software-Security
NFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
NFO: More about the report processing at http://localhost:9000/api/ce/task?id=faed29a7-e9f6-4778-8013-938e03f1b909
NFO: Analysis total time: 2:47.177 s
NFO: ------------------------------------------------------------------------
NFO: EXECUTION SUCCESS
NFO: ------------------------------------------------------------------------
NFO: Total time: 2:48.577s
NFO: Final Memory: 41M/140M
NFO: ------------------------------------------------------------------------
```

Figure 6: Analysis details

Once the analysis was finished, the local server updated the information and provided it with an overview, issues, security hotspots, the measures we can take and information about code.

The overall summary of the analysis was provided in the overview section as shown in Figure 7. There was a total of 4 reliability issues, 363 maintainability issues, and 37 security issues.


Security
0 Open issues A
0 H 0 M 0 L

Reliability
4 Open issues E
3 H 1 M 0 L

Maintainability
363 Open issues A
149 H 94 M 120 L

Accepted issues
0
Valid issues, but not fixed. They represent accepted technical debt.

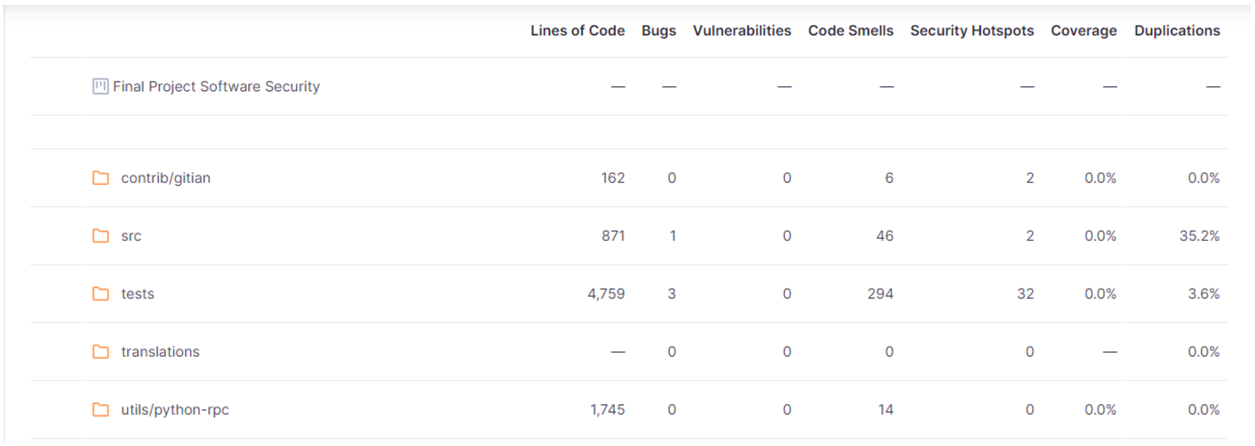
Coverage
0.0%
On 6k New Lines to cover.

Duplications
0.2%
On 298k New Lines.

Security Hotspots
37 E

Figure 7: Summary of the analysis

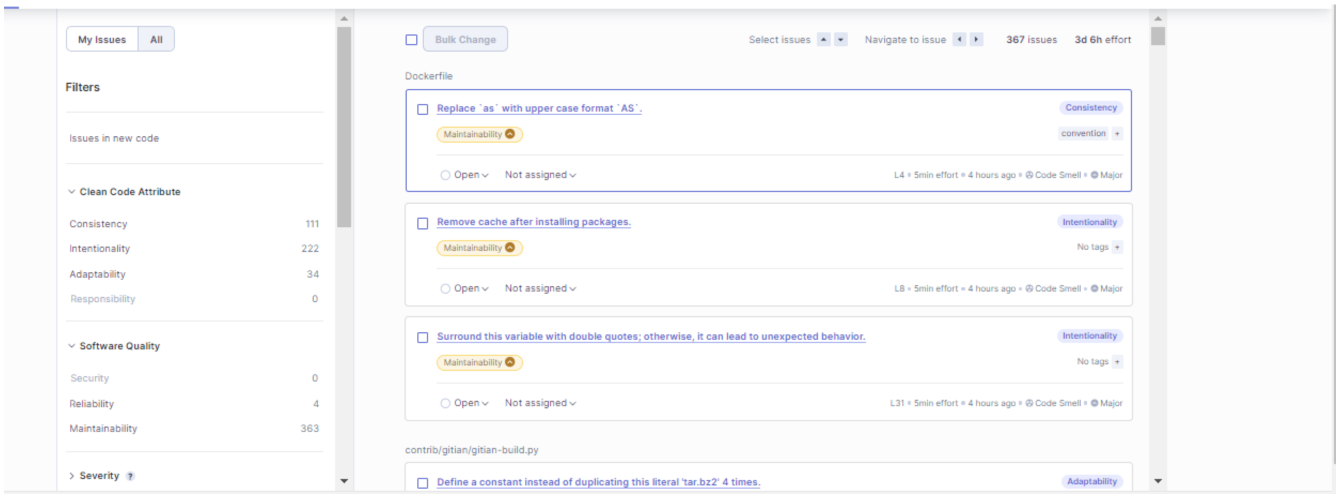
Let's start with the breakdown of the code. After the analysis, SonarQube provided the code present in each file, the number of lines, bugs, vulnerabilities, security issues, code smell and duplications in each file as shown in Figure 8.



	Lines of Code	Bugs	Vulnerabilities	Code Smells	Security Hotspots	Coverage	Duplications
Final Project Software Security	—	—	—	—	—	—	—
contrib/gitian	162	0	0	6	2	0.0%	0.0%
src	871	1	0	46	2	0.0%	35.2%
tests	4,759	3	0	294	32	0.0%	3.6%
translations	—	0	0	0	0	—	0.0%
utils/python-rpc	1,745	0	0	14	0	0.0%	0.0%

Figure 8: Code information

Other than the breakdown of code, it also provided us with the in-depth description of each problem with information like where the problem is, why is that a problem, and how we can fix it as shown in figure 9 and 10.



My Issues | All

Filters

Issues in new code

Clean Code Attribute

Consistency	111
Intentionality	222
Adaptability	34
Responsibility	0

Software Quality

Security	0
Reliability	4
Maintainability	363

Severity

Bulk Change   Select issues   Navigate to issue   367 issues   3d 6h effort

Dockerfile

Replace `as` with upper case format `AS`.   Consistency
Maintainability   convention
Open   Not assigned   L4 • 5min effort • 4 hours ago • Code Smell • Major

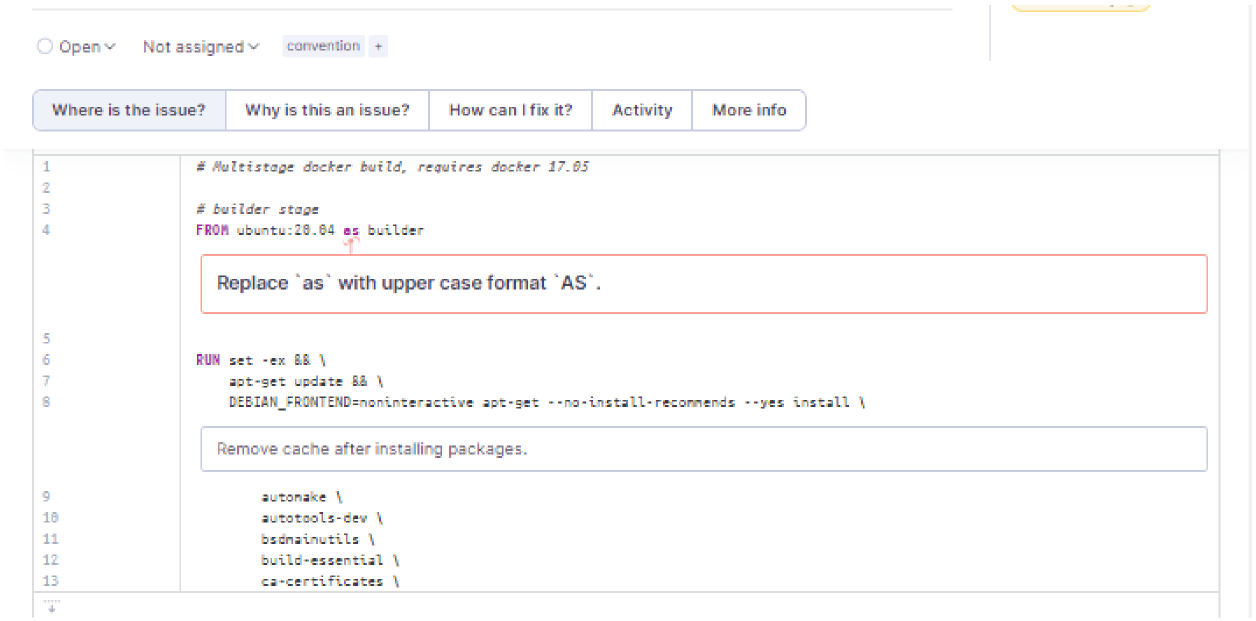
Remove cache after installing packages.   Intentionality
Maintainability   No tags
Open   Not assigned   L8 • 5min effort • 4 hours ago • Code Smell • Major

Surround this variable with double quotes; otherwise, it can lead to unexpected behavior.   Intentionality
Maintainability   No tags
Open   Not assigned   L31 • 5min effort • 4 hours ago • Code Smell • Major

contrib/gitian/gitian-build.py

Define a constant instead of duplicating this literal 'tar.bz2' 4 times.   Adaptability

Figure 9: In-depth detail of the issues



Open   Not assigned   convention

Where is the issue? | Why is this an issue? | How can I fix it? | Activity | More info

```
# Multistage docker build, requires docker 17.05

# builder stage
FROM ubuntu:20.04 as builder
```

Replace `as` with upper case format `AS`.

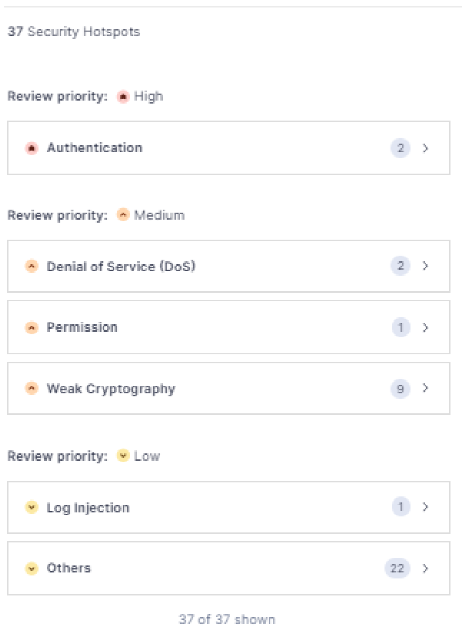
```
RUN set -ex && \
    apt-get update && \
    DEBIAN_FRONTEND=noninteractive apt-get --no-install-recommends --yes install \
```

Remove cache after installing packages.

```
        automake \
        autotools-dev \
        bsdmainutils \
        build-essential \
        ca-certificates \
```

Figure 10: Different sections containing information about the issue

Since the focus of the project is to analyze the code and find the vulnerabilities in it, we will be only working on the security issues found in the source code. As discussed before, there are 37 security hotspots found in this system from which 2 are of high priority, 12 are of medium severity, and 23 of them are of low priority as shown in Figure 11.


37 Security Hotspots

Review priority: High

Authentication	2

Review priority: Medium

Denial of Service (DoS)	2
Permission	1
Weak Cryptography	9

Review priority: Low

Log Injection	1
Others	22

37 of 37 shown

Figure 11: Details of the security hotspot

We will be elaborating on the following list of vulnerabilities:

1. Authentication (High)
2. Denial of Service (Medium)
3. Weak Cryptography (Low)
4. Log Injection (Low)

3.1 Authentication:

One of the high priority security issues we found is authentication issue which is due to a potentially hard-coded credential as seen in figure 12. This can lead to many vulnerabilities allowing attackers to access sensitive information very easily. In OWASP 10 2021 list, this kind of vulnerability has been identified with A07: Identification and Authentication Failures. Moreover, it has led to the following attacks:

CVE-2018-15389:

A vulnerability in the install function of Cisco Prime Collaboration Provisioning (PCP) had default hard-coded username and password[1]. When this password isn't changed, it allows the attacker to access administrator-level privileges, hence, compromising user information.

CVE-2019-13466:

Western Digital SSD Dashboard before 2.5.1.0 and SanDisk SSD Dashboard before 2.5.1.0 have Incorrect Access Control [2]. The “generate reports” archive is protected with a hard-coded password.

The vulnerability caused due to such a mistake can be found in CWE articles: CWE-798: Use of Hard-coded Credentials and CWE-259: Use of Hard-coded Password.

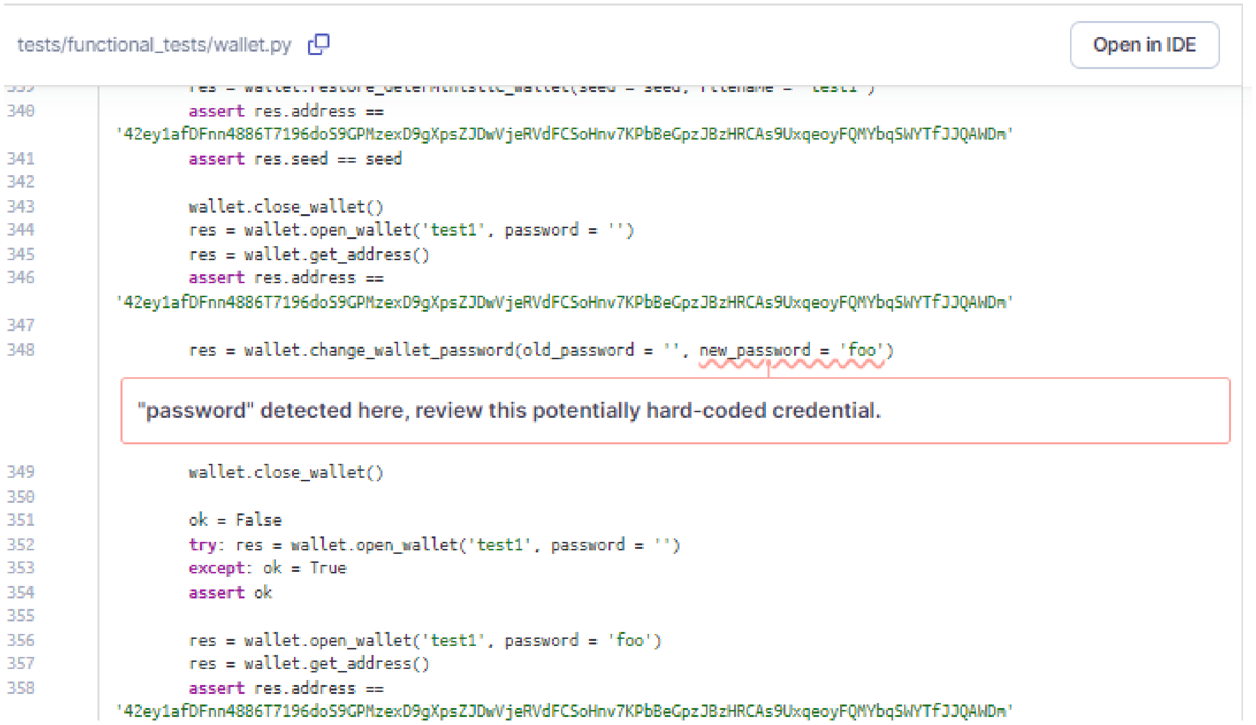
CWE-798: Use of Hard-coded Credentials:

The product includes static credentials, like passwords or cryptographic keys, which it employs for internal authentication, communication with external components, or data encryption [3]. The hard-coded credentials configured by the admin can become an open invitation for an attacker to get into the system or product. For in-bound variant, there is an administration default password that cannot be changed without manually modifying the program, whereas outbound variant applies to front-end systems that authenticate with a back-end service. This means the back end has a fixed password that can be accessed by any user. This weakness is caused during the architecture and design phase of the product.

Potential mitigations are:

- For outbound authentication, securely store passwords, keys, and credentials outside of the code in an encrypted configuration file or database, inaccessible to unauthorized users, with stringent permissions or leverage tools like Windows Encrypted File System (EFS) for protection.
- For inbound: let the user decide their unique password on first login mode.
- If hard-coded passwords cannot be removed, keep access control checks at most of the points.
- Apply strong one-way hashing so that if an attacker tries to enter a password, the hash can be compared to the stored hash.
- For front-end to back-end connections, the passwords can be automatically changed in time intervals and passwords should be limited to back-end only, the user shouldn’t have full access.

CWE-259: Use of Hard-coded Password is just like CWE-798 as described above. These articles emphasize the severity of this vulnerability which is very high because attackers can very easily exploit this vulnerability and then threaten the sensitive information of the user. Hence, hardcoded passwords should be avoided at every cost or there should be some alternative that does not allow an attacker to hack into the system.


tests/functional_tests/wallet.py

Open in IDE

```
        res = wallet.restore_deterministic_wallet(seed = seed, filename = 'test1')
        assert res.address ==
'42ey1afDFnn4886T7196doS9GPMzexD9gXpsZJDwVjeRVdFCSoHnv7KPbBeGpzJBzHRCAs9UxqeoyFQMYbqSWYTfJJQAWDm'
        assert res.seed == seed

        wallet.close_wallet()
        res = wallet.open_wallet('test1', password = '')
        res = wallet.get_address()
        assert res.address ==
'42ey1afDFnn4886T7196doS9GPMzexD9gXpsZJDwVjeRVdFCSoHnv7KPbBeGpzJBzHRCAs9UxqeoyFQMYbqSWYTfJJQAWDm'

        res = wallet.change_wallet_password(old_password = '', new_password = 'foo')
```

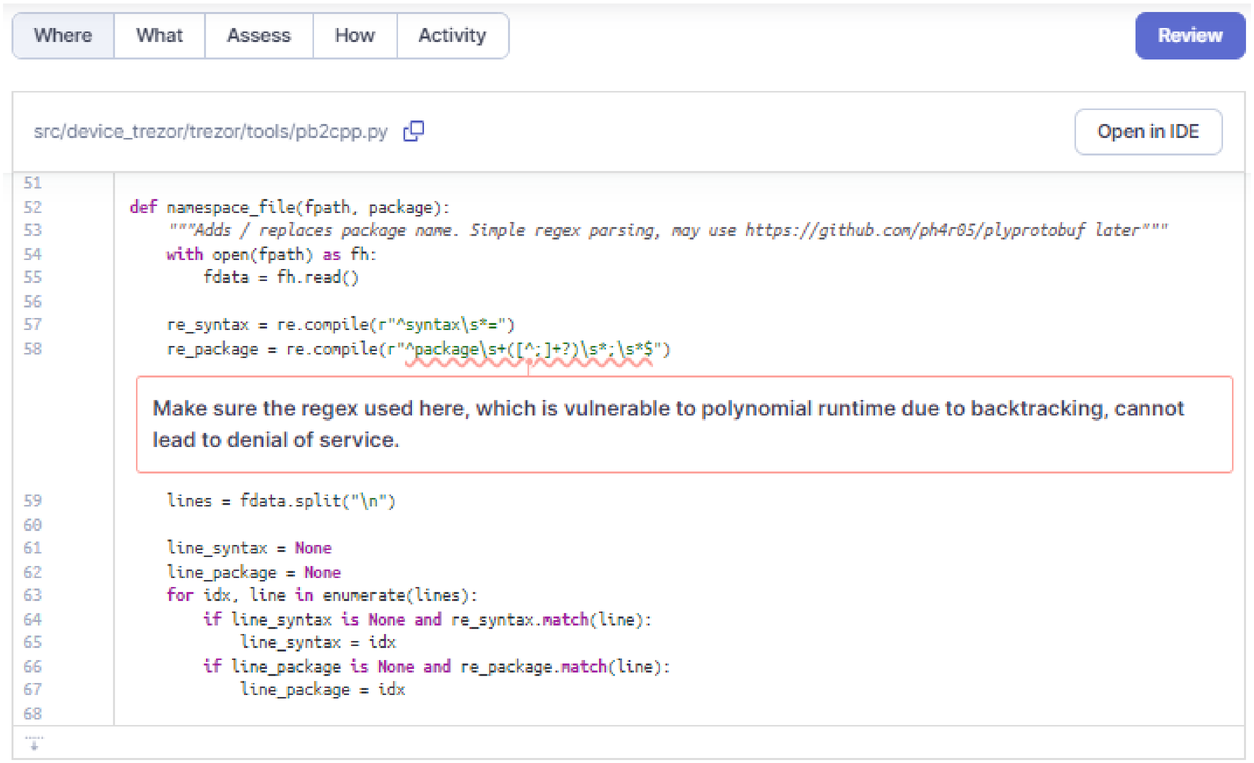
"password" detected here, review this potentially hard-coded credential.

```
        wallet.close_wallet()

        ok = False
        try: res = wallet.open_wallet('test1', password = '')
        except: ok = True
        assert ok

        res = wallet.open_wallet('test1', password = 'foo')
        res = wallet.get_address()
        assert res.address ==
'42ey1afDFnn4886T7196doS9GPMzexD9gXpsZJDwVjeRVdFCSoHnv7KPbBeGpzJBzHRCAs9UxqeoyFQMYbqSWYTfJJQAWDm'
```

Figure 12: code snippet that identifies the hard-coded credential.

3.2 Denial of Service (DOS):

After the analysis, we found 12 medium severity security issues out of which 2 were of DOS. Before identifying the cause of this attack, it is important to understand what DOS is. The Regular expression Denial of Service (ReDoS) is a Denial of Service attack, that exploits the fact that most Regular Expression implementations may reach extreme situations that cause them to work very slowly. An attacker can then cause a program using a Regular Expression to enter these extreme situations and then hang for a very long time [4]. Now, in this case the denial-of-service attack is caused due to a regular expression that was written on line 58 of a function called "namespace_file" present in "src/device_trezor/trezor/tools/pb2cpp.py" as shown in Figure 13.



src/device_trezor/trezor/tools/pb2cpp.py

```python
def namespace_file(fpath, package):
    """Adds / replaces package name. Simple regex parsing, may use https://github.com/ph4r05/plyprotobuf later"""
    with open(fpath) as fh:
        fdata = fh.read()

    re_syntax = re.compile(r"^syntax\s*=")
    re_package = re.compile(r"^package\s+([^;]+?)\s*;\s*$")
```

Make sure the regex used here, which is vulnerable to polynomial runtime due to backtracking, cannot lead to denial of service.

```python
    lines = fdata.split("\n")

    line_syntax = None
    line_package = None
    for idx, line in enumerate(lines):
        if line_syntax is None and re_syntax.match(line):
            line_syntax = idx
        if line_package is None and re_package.match(line):
            line_package = idx
```

Figure 13: Code snippet that identifies denial-of-service

Basically, the regular expression written in this line causes catastrophic backtracking. Most of the regular expressions have a backtracking engine which helps them to try all possible paths of the regular expression, but in worst cases the time complexity of a regular expression is exponential in size and can cause catastrophic backtracking, which ultimately leads to denial-of-service. There is a type of regular expression called the “Evil Regex”, which is an expression that gets stuck on a crafted input. In the case where regular expression is dependent on the input, the attacker can take advantage of it and craft an evil regex which would make the system vulnerable. In OWASP 10 2017 list[5], this kind of vulnerability has been identified with A1:2017-Injection.

CVE-2024-21490:

After going over the list of CVE articles, I came across a similar vulnerability that was mentioned in CVE-2024-21490[6]. This vulnerability was found in the package "angular" version 1.3.0 and was also caused due to the run time of the expression causing catastrophic backtracking. According to the NVD article, this vulnerability has a CVSS of 7.5 and is considered as a high severity vulnerability.

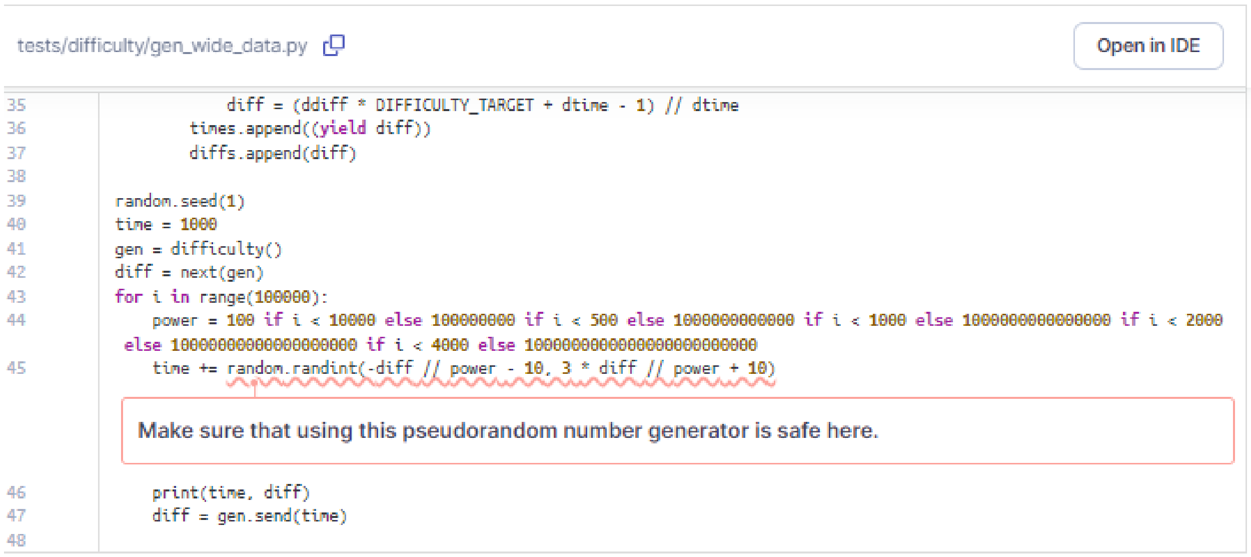
CWE-1333: Inefficient Regular Expression Complexity

The CWE article CWE-1333: Inefficient Regular Expression Complexity tells us more about vulnerability [7]. Certain regular expression engines employ a mechanism known as "backtracking," where if a token fails to match, the engine retreats to a prior position to attempt a different token. However, backtracking can pose a vulnerability under specific conditions: when the potential backtracking attempts grow exponentially with input length, when the input fails to match the regular expression, and when the input is sufficiently lengthy. Exploiting this vulnerability, attackers can create tailored inputs designed to trigger excessive backtracking, leading to a spike in CPU consumption.

Some of the possible mitigations are:

- To use regular expressions that do not allow back tracking.
- To set limit of the back tracking in the regular expression's configuration.
- Avoid using regular expressions that are not trusted. If any expression is used, make sure that it avoids back tracking.
- Limit the length of the input that the regular expression will process.

3.3 Weak Cryptography:



```
            diff = (ddiff * DIFFICULTY_TARGET + dtime - 1) // dtime
        times.append((yield diff))
        diffs.append(diff)

random.seed(1)
time = 1000
gen = difficulty()
diff = next(gen)
for i in range(100000):
    power = 100 if i < 10000 else 100000000 if i < 500 else 1000000000000 if i < 1000 else 1000000000000000 if i < 2000
 else 10000000000000000000 if i < 4000 else 1000000000000000000000000
    time += random.randint(-diff // power - 10, 3 * diff // power + 10)
```

Make sure that using this pseudorandom number generator is safe here.

```
    print(time, diff)
    diff = gen.send(time)
```

Figure 14: Code snippet that shows where the pseudorandom number is generated.

In figure 14, there is a random number generated that can be predictable and it can lead to many vulnerabilities. These pseudorandom number generators can be security-sensitive and can allow the attacker to easily predict the number generated. This can be further explained by these attacks related to pseudorandom generated numbers:

CVE-2013-6386:

Drupal 6.x before 6.29 and 7.x before 7.24 uses the PHP mt_rand function to generate random numbers, which uses predictable seeds and allows remote attackers to predict security strings and bypass intended restrictions via a brute force attack [8].

CVE-2006-3419:

Tor before 0.1.1.20 uses OpenSSL pseudo-random bytes (RAND_pseudo_bytes) instead of cryptographically strong RAND_bytes, and seeds the entropy value at start-up with 160-bit chunks without reseeding, which makes it easier for attackers to conduct brute force guessing attacks [9].

CVE-2008-4102:

Joomla! 1.5 before 1.5.7 initializes PHP's PRNG with a weak seed, which makes it easier for attackers to guess the pseudo-random values produced by PHP's mt_rand function, as demonstrated by guessing password reset tokens, a different vulnerability than CVE-2008-3681 [10].

These attacks tell us that attackers are capable of brute-force guessing these numbers, hence, they are successful at getting into the system and put it at risk. Therefore, we need to understand which vulnerability is caused due to the usage of pseudorandom generated numbers and how it can be mitigated.

The CWE articles addressing this vulnerability are:

CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG):

Using a non-specialized random number generator for cryptography can be risky. These generators aren't built to be secure for encryption [11]. While they might be okay for some purposes, they can make cryptography vulnerable to certain attacks. They're often weaker and less secure compared to specialized generators, but they might be used because they're faster and don't use up as much valuable resources. However, their weaknesses could be exploited to break the encryption. This vulnerability is caused during the architecture and design phase.

There is only one potential mitigation mentioned which is to utilize functions or hardware that employ hardware-based random number generation for all cryptographic operations. This approach is the preferred solution, utilizing CryptGenRandom on Windows or hw_rand() on Linux.

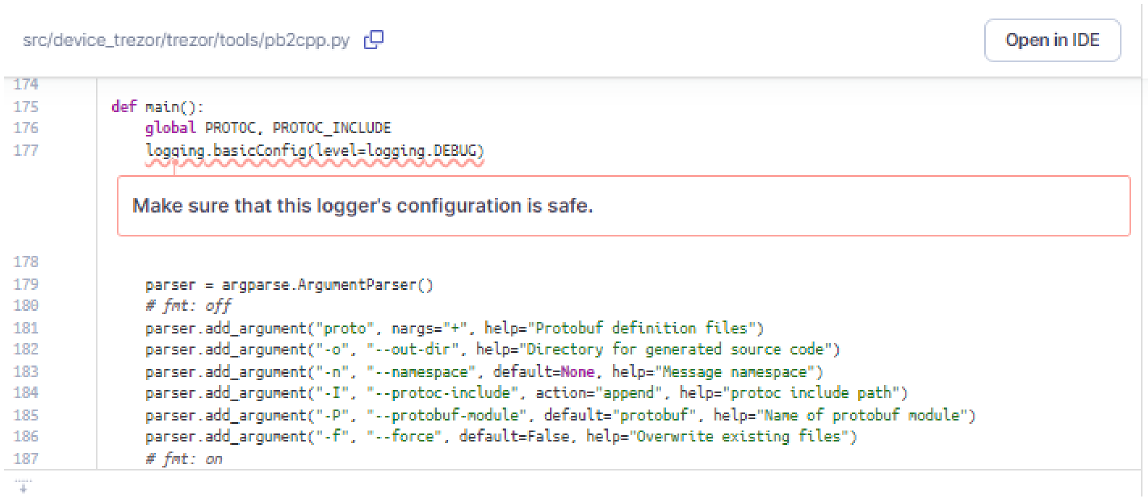
CWE-326: Inadequate Encryption Strength:

The product uses an encryption method that is not quite strong. Therefore, the system can be subjected to brute force attacks that can lead to successful exploitation by an attacker [12]. This vulnerability is caused due to a design and architecture problem. The potential mitigation for this vulnerability is to use a stronger encryption method.

This vulnerability identified in Monero is of low severity and it can be addressed by using stronger encryption methods and a true random number generator instead of relying on pseudorandom number generators. We think that this vulnerability is of low severity as it requires the attacker to put in more work when it comes to brute forcing their guesses. Hence, it might take them more time and effort to exploit this vulnerability. However, it is still important to have the relevant defense mechanisms so that the system is potentially secure.

3.4 Log injection:

One of the low-severity security issues found in Monero is Log injection. This vulnerability was found at "src/device_trezor/trezor/tools/pb2cpp.py", exactly on the line number 177 as shown in Figure 15.



```
174
175     def main():
176         global PROTOC, PROTOC_INCLUDE
177         logging.basicConfig(level=logging.DEBUG)
```

Make sure that this logger's configuration is safe.

```
178
179         parser = argparse.ArgumentParser()
180         # fmt: off
181         parser.add_argument("proto", nargs="+", help="Protobuf definition files")
182         parser.add_argument("-o", "--out-dir", help="Directory for generated source code")
183         parser.add_argument("-n", "--namespace", default=None, help="Message namespace")
184         parser.add_argument("-I", "--protoc-include", action="append", help="protoc include path")
185         parser.add_argument("-P", "--protobuf-module", default="protobuf", help="Name of protobuf module")
186         parser.add_argument("-f", "--force", default=False, help="Overwrite existing files")
187         # fmt: on
```

Figure 15: Code snippet that identifies Log injection

As we know, almost every application and device maintain a log to track the activity of the user, and in case of attack helps us to identify what degree of damage the attacker has done. Hence, we can say that logs are important before, during, and after any security incident. However, logs can also be a target of the attack as they might contain sensitive information like passwords, API keys, or other confidential information. In this case, the code written on line # 177 is causing the log to write sensitive information as well.

Configuring the logs is security sensitive and has caused many vulnerabilities in the past as discussed below:

CVE-2018-0285:

CVE-2018-0285 [13] which describes a vulnerability that occurred at Cisco Prime Service Catalog due to logging. Here, the attacker exploited this vulnerability by performing certain operations that caused excessive logging which ultimately led to exhaustion of disk space due to which it denied the service to its user.

CVE-2000-1127:

Another similar case is mentioned in CVE-2000-1127 [14] which tells about the vulnerability found in HP resource monitor service. This service allowed the users to read and edit the log files, rename it and create a symbolic link to the target file, due to which the log became public.

CVE-2017-15113:

CVE-2017-15113[15] describes a vulnerability found in ovirt-engine before version 4.1.7.6 which included passwords with masking in the logs making it readable to the vendors and the other parties with which the logs were shared.

After going through these cases, it is quite clear how this vulnerability can be exploited and cause harm. In OWASP 10 2021 [16] list, this kind of vulnerability has been identified with A09:2021 – Security Logging and Monitoring Failures.

Potential mitigations recommended by SonarQube for this vulnerability are:

- To store these logs in a secure place so that only authorized personnels can access them.
- Check that your production deployment doesn’t have its loggers in "debug" mode as it might write sensitive information in logs.
- Check that the permissions of the log files are correct. If you index the logs in some other service, make sure that the transfer and the service are secure too.
- Add limits to the size of the logs and make sure that no user can fill the disk with logs.
- Choose log format which is easy to parse and process automatically. It is important to process logs rapidly in case of an attack so that the impact is known and limited.
- Configure the loggers to display all warnings, info and error messages.

CWE 532: Insertion of Sensitive Information into Log File:

The CWE article 532: Insertion of Sensitive Information into Log File[18] tells that although logging is an important step in developing stages, it should be removed before the product is shipped as it can accidentally expose sensitive to attackers. Some of the potential mitigations provided in this article are:

- Do not write sensitive information in the log files.

- Remove debug log files before deploying the application into production.
- Protect log files against unauthorized read/write.

CWE-117: Improper Output Neutralization for Logs:

Another CWE article, CWE-117: Improper Output Neutralization for Logs [19] tells us how the logging vulnerability can be exploited. This occurs when an attacker gains access to the log files and injects malicious content into it. To avoid such instances, the following measures should be taken:

- Assume all inputs are malicious and make a standard format for the log input.
- Before validating inputs received by an application, they should undergo two processes: decoding and canonicalization as by doing so the application can effectively and securely process user input, reducing the risk of vulnerabilities such as injection attacks or data manipulation.

4. Attack Scenarios:

4.1 Authentication:

- **Scenario:** A user downloads the open-source code of the project and then figures out a way to access the hard-coded password required for admin access. Therefore, the user accesses the admin privileges and makes malicious changes to the software.
- **Mitigation Plan:** For out-bound authentication, passwords should be kept encrypted outside of the code in another file and it should be inaccessible to unauthorized users. Moreover, in design phase this can be catered to by "Going Least Privilege". Security controls should be applied to keep access control.

4.2 Denial of Service:

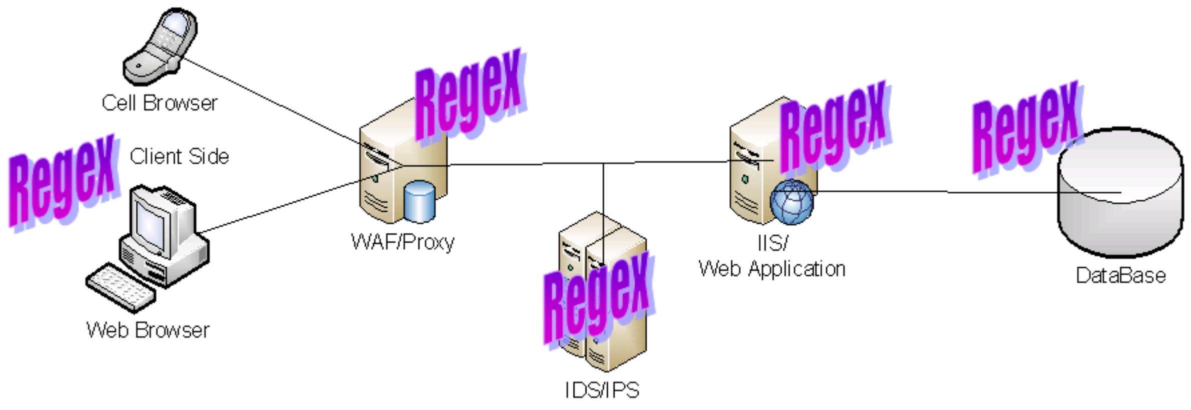


Figure 16: Regex in every layer of web

- **Scenario:** As seen in Figure 16, regular expression is present in every layer of web[1], and hence the attackers can take advantage of this situation, since the code contains a regular expression that runs in exponential time, the input given to the regular expression by the attacker can make the program run for a long period of time. Hence, utilizing CPU resources due to which the program can no longer accept user requests resulting in Denial of Service (DOS).
- **Mitigation Plan:** A regular expression is causing denial-of-service which is dependent on the input given by the user. The attacker can use this point and can give an input of a large length, hence, to avoid this we can set a limit on the input by restricting the length and complexity. Other than that, we can implement strict input validation to prevent malicious input to reach to the regular expression.

4.3 Weak Cryptography:

- **Scenario:** The code has a pseudorandom number generator so when this generator generates an important token for the user, the attacker can easily guess this token by brute force guessing and then accessing sensitive information before the user can.
- **Mitigation Plan:** To avoid such a threat, the code should use a true random number generator. Moreover, in the design phase, this can be catered to by implementing defense in depth and strong access control should be enforced. The tokens generated can also be made longer so it is difficult for the attacker to brute force.

4.4 Log injection:

- **Scenario:** Armed with the sensitive information obtained from the logs, the attacker attempts to gain unauthorized access to the system. For example, they may use database credentials to access the database directly or use API keys to make unauthorized requests to external services.
- **Mitigation Plan:** In this case, the attacker can access sensitive information from the logs because the code has set the log level to “Debug”, which should only be during the development of the product. Hence, the first step to remove this vulnerability is to change the log level from “Debug” to “Info”, “Error”, or “Warning” according to the environment of the application. Another thing that can be done is to identify the sensitive information and instead of directly storing it in logs, we should create a placeholder for them, or we can mask them properly such that the attacker can’t guess it.

5. Threat Model:

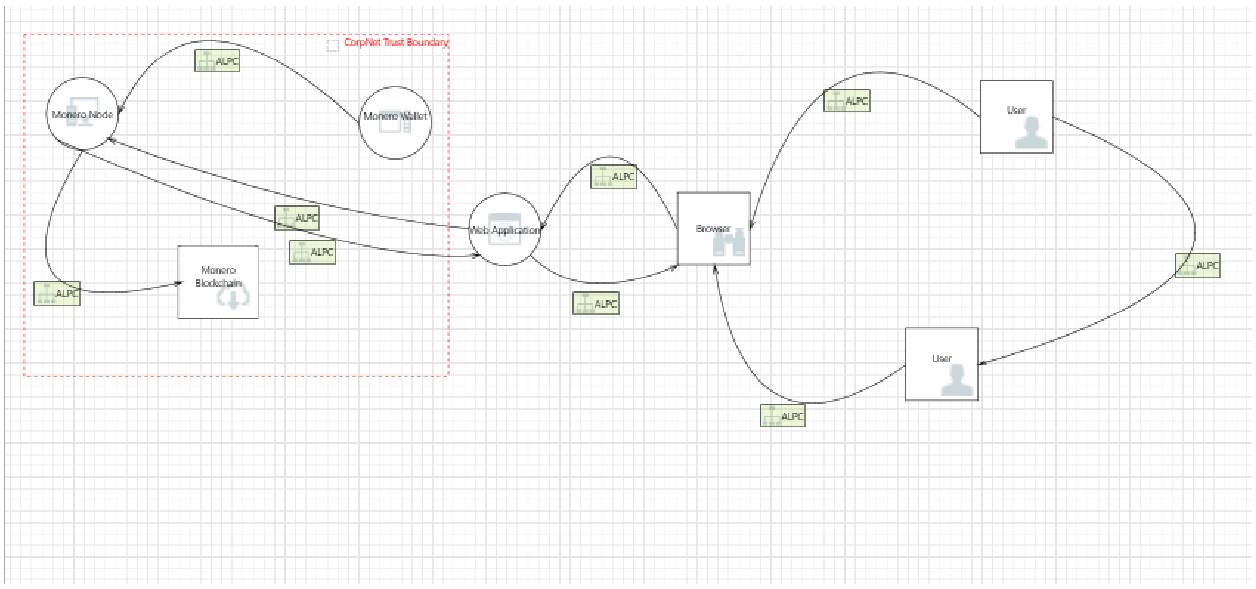


Figure 17: Threat modeling of Monero

A threat model for software is a structured approach to identifying and prioritizing potential threats to the security of the software system. It involves analyzing the system's architecture, design, and implementation to anticipate potential vulnerabilities and threats that could exploit them. To create a threat model, we first identify the assets and then design the flow of information. After that we can identify the threats and assess the vulnerabilities.

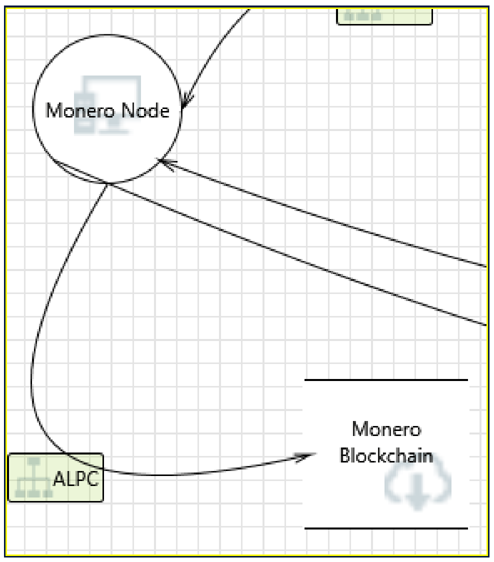
For this project, we used Microsoft Threat Modeling tool which we downloaded from this link.

In this model, we have several components that make up Monero. The Monero node as seen in Figure 17 is the main server and the Monero wallet is used to store, send and receive Monero points, hence, it is the most important component for transactions. Then, there is the Monero blockchain component that stores all the important information related to the users and their transactions. We have shown which components are at the same trust level using the Corp Net boundary. This allows us to differentiate between trusted internal components of the Monero ecosystem and external entities, helping to identify areas where trust assumptions can be made, and potential security boundaries exist. In this boundary, we have three main Monero components: Monero Wallet, Monero Blockchain, Monero Node.

To show how the user interacts with Monero, the user first sends their request to the browser to make a transaction. The browser validates the user request and checks with the Monero Blockchain to accept it. Then, the Monero Blockchain communicates with the Monero Wallet to perform the transaction and the Monero Node ensures security and integrity during the transaction. After both the users' details have been verified, a secure transaction is made.

Some of the vulnerabilities that we identified in our threat model using the tool are:

Interaction: ALPC



1. **Spoofing of Destination Data Store Monero Blockchain:**
- Category: Spoofing

Description: This is a high priority vulnerability as the attacker can modify the data by writing to the Monero Blockchain. A strong standard authentication mechanism can help here. As identified above in our vulnerabilities section from SonarQube, authentication remains an issue because of the hard-coded password which can provide the attacker with admin privileges and threaten the system.

2. **Data Flow ALPC Is Potentially Interrupted:**
- Category: Denial of Service

Description: The data going from web application to the Monero Node can be interrupted by an attacker.

3. **Data Flow Sniffing:**

Category: Information Disclosure

Description: Data flow can be sniffed by an attacker which means that they can check what type of data is flowing and then exploit it in several malicious ways. For this, strong encryption of data is necessary.

4. **Potential Lack of Input Validation for Monero Node:**

- Category: Tampering

Description: Data flowing across ALPC may be tampered with by an attacker. This may lead to a denial-of-service attack against Monero Node or an elevation of privilege attack against Monero Node or an information disclosure by Monero Node. Failure to verify that input is as expected is a root cause of many exploitable issues. Consider all paths and the way they handle data. Verify that all input is verified for correctness using an approved list input validation approach.

These are some of the vulnerabilities we found interesting in our design of the threat model. To reiterate, this is just a basic structure based on our understanding of the software. We wanted to try threat modeling on our own and we learned a lot. There were several parts of the tools that we had not yet discovered before this. Therefore, it was an insightful experience and generating a report from our own threat model was very fascinating.

6. Feedback:

In this assignment, we used SonarQube which was a new static analysis tool for us. It was also a very convenient tool to use as it gave analysis very quickly and in a lot of detail as compared to MobSF. There was a lot of information regarding the code that was very useful for us, and the issues identified were divided into separate sections. That made it easy for us to write about the vulnerabilities identified. Through this assignment, we also discovered a new cryptocurrency software. Moreover, we did not know regular expressions that led to denial-of-service vulnerability, so that was new for us to learn. Lastly, we used the Microsoft threat modeling tool to create a model on our own which was a good learning experience for us.

References:

[1] National Vulnerability Database (NVD). (2018). CVE-2018-15389. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-15389 (L)

[2] National Vulnerability Database (NVD). (2019). CVE-2019-13466. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-13466 (L)

[3] MITRE Corporation. (n.d.). CWE-798: Use of Hard-coded Credentials. [Online]. Available: https://cwe.mitre.org/data/definitions/798.html (L)

[4] OWASP, "Regular expression Denial of Service - ReDoS," OWASP, [Online]. Available: https://owasp.org/www-community/attacks/Regular_expression_Denial_of_Service_-_ReDoS.

[5] OWASP. (2017). OWASP Top Ten - 2017. [Online]. Available: https://owasp.org/www-project-top-ten/2017/Top_10

[6] Common Vulnerabilities and Exposures (CVE). (2024). CVE-2024-21490. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-21490

[7] Common Weakness Enumeration (CWE). (n.d.). CWE-1333: Improper Handling of Excessive Resource Consumption ('Resource Exhaustion'). [Online]. Available: https://cwe.mitre.org/data/definitions/1333.html

[8] National Vulnerability Database (NVD). (2013). CVE-2013-6386. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2013-6386

[9] National Vulnerability Database (NVD). (2006). CVE-2006-3419. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2006-3419

[10] National Vulnerability Database (NVD). (2008). CVE-2008-4102. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2008-4102

[11] MITRE Corporation. (n.d.). CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG). [Online]. Available: https://cwe.mitre.org/data/definitions/338.html

[12] MITRE Corporation. (n.d.). CWE-326: Inadequate Encryption Strength. [Online]. Available: https://cwe.mitre.org/data/definitions/326.html

[13] Common Vulnerabilities and Exposures (CVE). (2018). CVE-2018-0285. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-0285

[14] Common Vulnerabilities and Exposures (CVE). (2000). CVE-2000-1127. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1127

[15] Common Vulnerabilities and Exposures (CVE). (2017). CVE-2017-15113. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-15113

[16] OWASP. (n.d.). OWASP Top Ten. [Online]. Available: https://owasp.org/Top10/

[18] Common Weakness Enumeration (CWE). (n.d.). CWE-532: Inclusion of Sensitive Information in Log Files. [Online]. Available: https://cwe.mitre.org/data/definitions/532.html

[19] Common Weakness Enumeration (CWE). (n.d.). CWE-117: Improper Output Neutralization for Logs. [Online]. Available: https://cwe.mitre.org/data/definitions/117.html