CHAMELEON

FOR OUR SMARTER WORLD

Static Application Security Testing (SAST)

Name: Ahmad Rahman
ID: 222035606

Contents

Executive summary:

This report presents the findings and insights from a Static Application Security Testing (SAST) conducted on the website codes. Utilizing SonarQube, a leading tool for code quality analysis, the testing aimed to identify security vulnerabilities, reliability issues, and maintainability concerns within the project's codebase. The analysis successfully highlighted areas of excellence and opportunities for improvement.

Introduction:

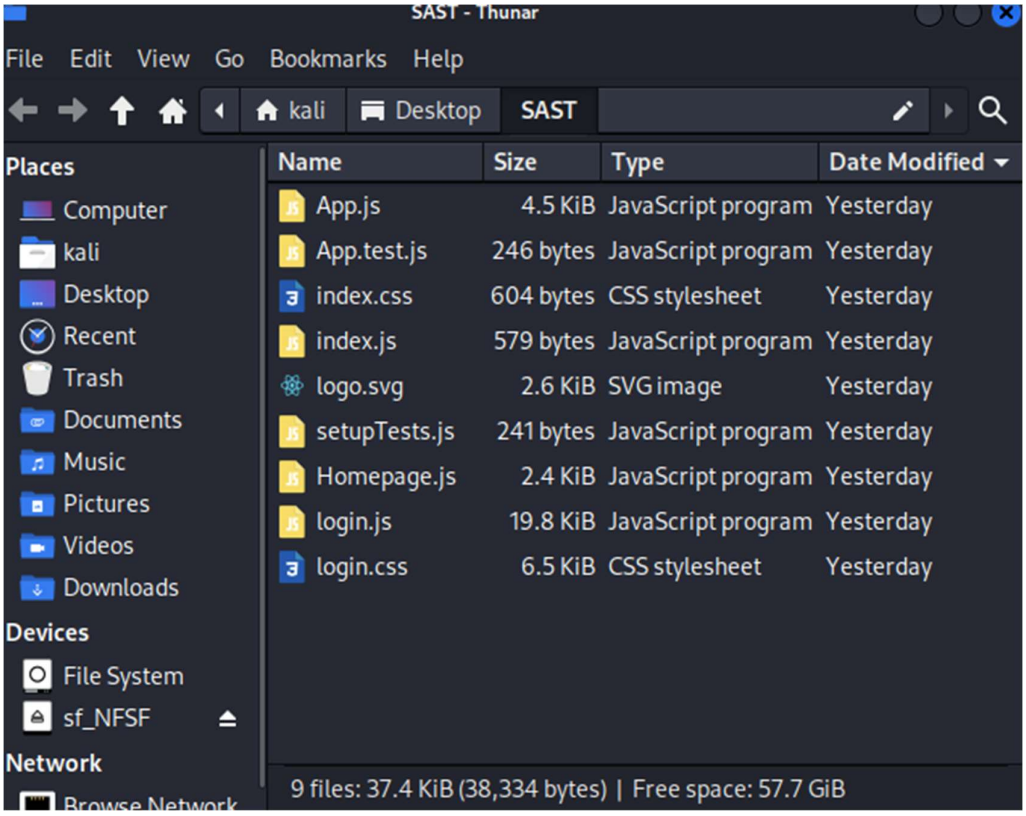
In this report, I will document my findings by SAST. What is SAST? Static Application Security Testing (SAST) is an essential process in software development that involves analysing source code to detect potential security vulnerabilities and coding flaws without running the program. The primary goal of SAST is to identify vulnerabilities early in the development cycle, making it easier and less costly to address issues. This report documents the SAST process carried out on the website, detailing the steps, tools used, findings, and recommended actions to enhance the code quality and security posture of the project.

Tools used:

For this analysis, we used SonarQube as the main tool. SonarQube is a well-known free platform that regularly checks the quality of code. It automatically reviews code to find bugs, security issues, and other problems in more than 20 programming languages. One of the key advantages of SonarQube is its detailed dashboard, which shows the health of the code clearly, helping developers easily spot and fix issues.

Scope of Testing

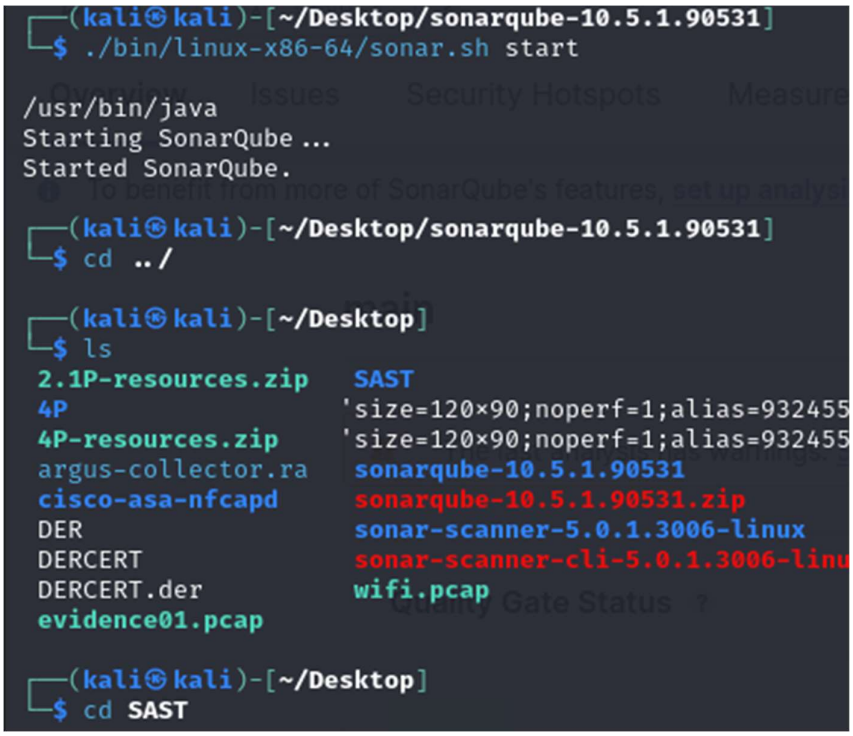
The scope of testing is the Chameleon website codes. I have taken some of the main source codes of the front end development and complied them into a folder.



Keeping these codes into one folder outside of GitHub makes it easier to run SonarQube scan in them

Steps and Results

To begin this, I first installed sonarqube. Then I started it on the following link http://localhost:9000



Running SonarQube, I logged into the server, and created a project.

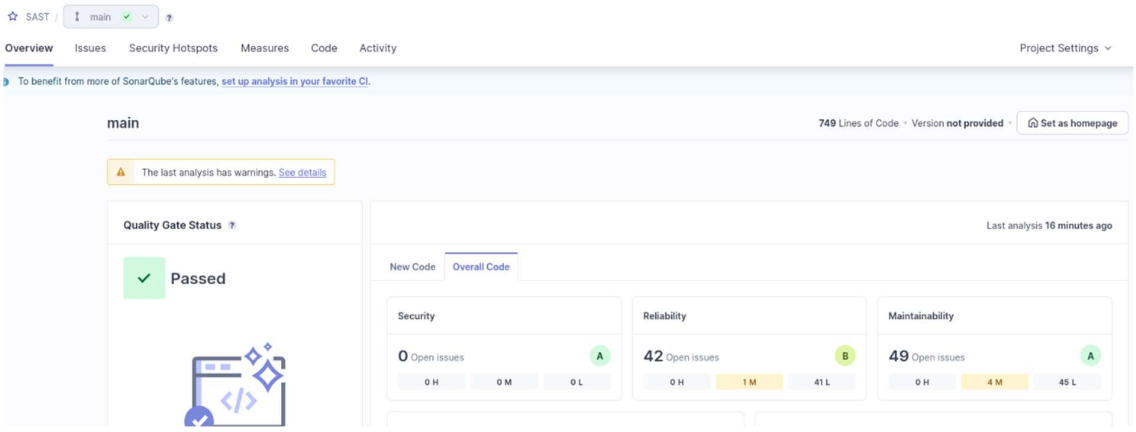
I had to download SonarScanner to complete the next task which I downloaded from a provided link in SonarQube

I chose to scan locally using the folders, I had to create a token key and then run the given command in the terminal inside the folder containing all the codes.

```
┌──(kali㉿kali)-[~/Desktop/SAST]
└─$ /home/kali/Desktop/sonar-scanner-5.0.1.3006-linux/bin/sonar-scanner \
  -Dsonar.projectKey=SAST \
  -Dsonar.sources=. \
  -Dsonar.host.url=http://localhost:9000 \
  -Dsonar.token=sqp_19bde4660f0b0892c6d5612b66ce286e438ec7ee
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
INFO: Scanner configuration file: /home/kali/Desktop/sonar-scanner-5.0.1.3006-linux/conf/sonar-scanner.properties
INFO: Project root configuration file: NONE
INFO: SonarScanner 5.0.1.3006
INFO: Java 17.0.7 Eclipse Adoptium (64-bit)
INFO: Linux 6.6.9-amd64 amd64
INFO: User cache: /home/kali/.sonar/cache
INFO: Analyzing on SonarQube server 10.5.1.90531
INFO: Default locale: "en_US", source code encoding: "UTF-8" (analysis is platform dependent)
INFO: Load global settings
INFO: Load global settings (done) | time=119ms
INFO: Server id: 147B411E-AY9mFAKevNjr1gtnY9uS
INFO: User cache: /home/kali/.sonar/cache
INFO: Loading required plugins
INFO: Load plugins index
INFO: Load plugins index (done) | time=91ms
INFO: Load/download plugins
INFO: Load/download plugins (done) | time=732ms
INFO: Process project properties
INFO: Process project properties (done) | time=10ms
INFO: Project key: SAST
INFO: Base dir: /home/kali/Desktop/SAST
INFO: Working dir: /home/kali/Desktop/SAST/.scannerwork
INFO: Load project settings for component key: 'SAST'
INFO: Load project settings for component key: 'SAST' (done) | time=47ms
INFO: Load quality profiles
INFO: Load quality profiles (done) | time=439ms
WARN: SCM provider autodetection failed. Please use "sonar.scm.provider" to define SCM of your project, or disable the SCM Sensor in the project se
INFO: Load active rules
INFO: Load active rules (done) | time=12735ms
INFO: Load analysis cache
INFO: Load analysis cache (404) | time=20ms
INFO: Preprocessing files...
INFO: 2 languages detected in 9 preprocessed files
INFO: Loading plugins for detected languages
INFO: Load/download plugins
INFO: Load/download plugins (done) | time=929ms
INFO: Inconsistent constructor declaration on bean with name 'org.sonarsource.scanner.api.internal.IsolatedClassloader@3e57cd70-org.sonar.scanner.i
uctor is effectively required since there is no default constructor to fall back to: public org.sonar.scanner.issue.IssueFilters(org.sonar.api.bat
INFO: Load project repositories
INFO: Load project repositories (done) | time=28ms
```

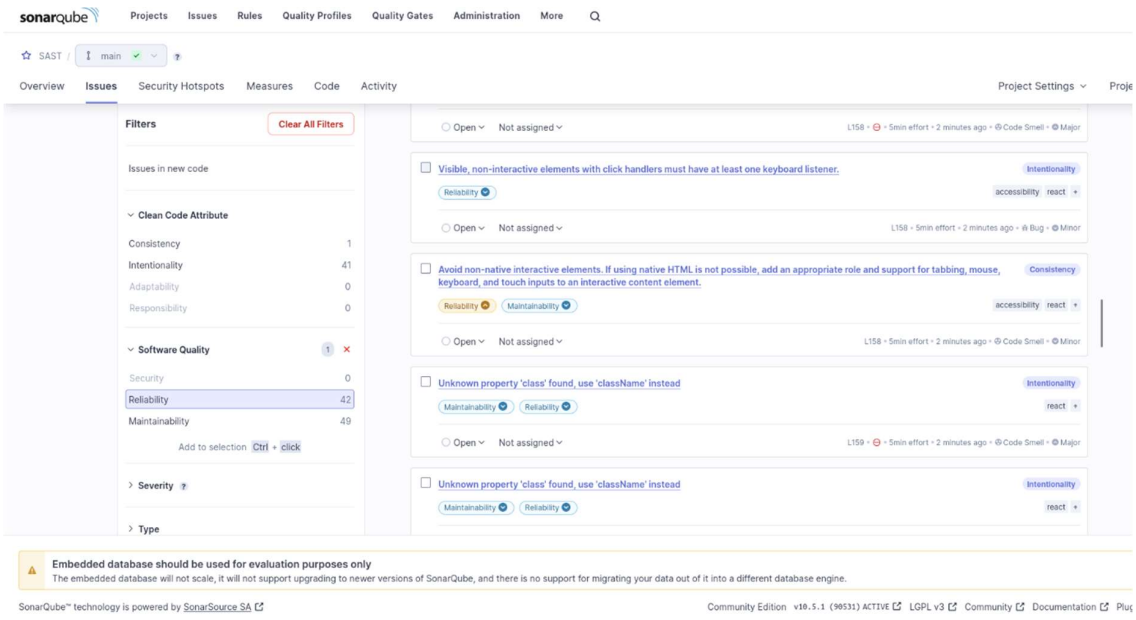
```
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 35.714s
INFO: Final Memory: 13M/54M
```

It was executed successfully and I could see the final outcome of the scan



Looking at the outcome, we can see that the results are quite good and we have passed.

Then I looked further into the results to see the scan better



There are some minor reliability issues however, they don't affect the system that much and aren't actually a defining reliability issue

The graph above shows the ratings given to them and it's A and B

SonarQube gives the above stating how it can be written better as well

Overall the scores were very good with majority A and one B

Recommendations

- Prioritize and Address Reliability Issues: Focus should be placed on addressing the medium severity reliability issue first, followed by systematic remediation of the low severity issues.
- Enhance Maintainability: Efforts should be made to resolve the medium severity maintainability issues, particularly those that contribute to code complexity and technical debt.
- Continuous Improvement: Integrate SonarQube scans into regular development cycles to continuously monitor and improve code quality. This practice will help in maintaining high standards and facilitating early detection of potential issues.
- Educational Focus: Developers should be encouraged to review the specific feedback from SonarQube to learn about best practices in coding and security, which can prevent similar issues in future projects.

Conclusion

In conclusion, the Static Application Security Testing (SAST) performed using SonarQube has provided valuable insights into the security, reliability, and maintainability of the website. The test results were highly encouraging, particularly in the area of security, where the project achieved an 'A' rating, indicating a strong security posture. However, the analysis also highlighted several areas where improvements are necessary, particularly in reliability, where 'B' ratings were received. This thorough examination revealed a total of 42 reliability issues and 49 maintainability issues. Addressing these issues is crucial for ensuring the application's stability and ease of future modifications. This proactive approach to software quality and security is essential in developing robust, efficient, and secure software systems.