College of Engineering and Physical Sciences

School of Informatics & Digital Engineering, Computer Science

Final Report

REPELSEC Security Testing Tool

Jonah Reader

Supervisor – Dr Paul Grace

March 2024

Declaration

Abstract

The objective of the REPELSEC project was to create a lightweight command-line security tool that can conduct Static Application Security Testing (SAST) and Software Composition Analysis (SCA) against a Java application. The tool was built to enhance the software security and reliability of Java projects by enabling developers to identify vulnerabilities and common bugs within the early stages of their Software Development Life Cycle (SDLC). The SAST component of the tool utilised static analysis techniques to test first-party source code for vulnerabilities that fall under categories such as sensitive data exposure, injection, and improper exception handling. The SCA component of the tool employed the National Institute of Standards and Technology's (NIST) National Vulnerability Database (NVD) Application Programming Interface (API) to assess if known vulnerabilities were associated with the third-party dependencies used in the project. To provide the user with feedback, the tool generated detailed reports in various file formats to highlight vulnerabilities and the recommended actions and resources to remediate them. The combined results from the SAST and SCA components demonstrated an effective and holistic approach to static vulnerability detection. Future perspectives involve incorporating advanced static analysis techniques to detect more complex vulnerabilities.

Keywords: Static Application Security Testing, Software Composition Analysis, Software security, Software Development Life Cycle, Security Tool

List of Acronyms

API	Application Programming Interface
AST	Application Security Testing
CI/CD	Continuous Integration/Continuous Deployment
CLI	Command Line Interface
CPE	Common Platform Enumeration
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
IDE	Integrated Development Environment
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
OWASP	Open Web Application Security Project
SAST	Static Application Security Testing
SCA	Software Composition Analysis
SDLC	Software Development Life Cycle
SSD	Secure Software Development
SSDLC	Secure Software Development Life Cycle
VSC	Visual Studio Code

Table of Contents

List of Figures

1 Introduction

1.1 Motivation

A significant problem that developers currently face is implementing the concepts of secure software engineering (software security) and Shift Left Security to create a Secure Software Development Life Cycle (SSDLC) process. Secure software engineering aims to avoid the appearance of security vulnerabilities by ensuring Secure Software Development (SSD) methods are employed at each stage of the SDLC. 'SSD methods include, among others, security specification languages, security requirements engineering processes, and software security assurance methods' (Khan and Zulkerine, 2009). Similarly, Shift Left Security is the idea of 'designing and testing for security early and continuously in each software iteration' (Pitchford, 2021). Shift Left Security has a heavy focus on preventing and remediating vulnerabilities early so that developers would not need to redesign their application due to architecture issues or retest their application after vulnerabilities have been remediated. By employing these concepts, developers are implementing a secure by design and security in depth strategy to create a SSDLC process in which security is the focus.

These concepts can be challenging to implement due to many Application Security Testing (AST) tools being time-consuming, complex to set up, and inconvenient to use for developers whose main priority is developing new features for an application. The REPELSEC tool contributes to solving this issue by giving developers the opportunity to test their code quickly and locally to find vulnerabilities at the earliest opportunity. This is achieved by enabling developers to conduct security analysis using a simple command-line interface (CLI) that tests files individually for quick and focused results. It also removes the need for files to be uploaded or analysis to be connected to a pipeline. Finally, the SAST and SCA functionality is also combined meaning separate tools are not needed for analysis.

REPELSEC can create many positive outcomes for a developer. Firstly, one outcome of using the tool is that developers write more secure and robust code due to them being able to receive quick and local feedback on present bugs and vulnerabilities. Secondly, because this tool is used during the coding stage of the SDLC, a developer will be more familiar with the codebase and technologies used for the project. This may allow them to more easily identify an issue and realise a solution faster with help from the tool. Finally, vulnerabilities will be found and potentially fixed before the code is pushed to a Continuous Integration/Continuous Deployment (CI/CD) pipeline for further testing or deployment. Figure 1 illustrates the price of fixing vulnerabilities at each stage of a SDLC. REPELSEC enables developers to test both early on and often during the coding stage, potentially preventing an organisation from spending more resources later into the SDLC to fix the vulnerabilities.

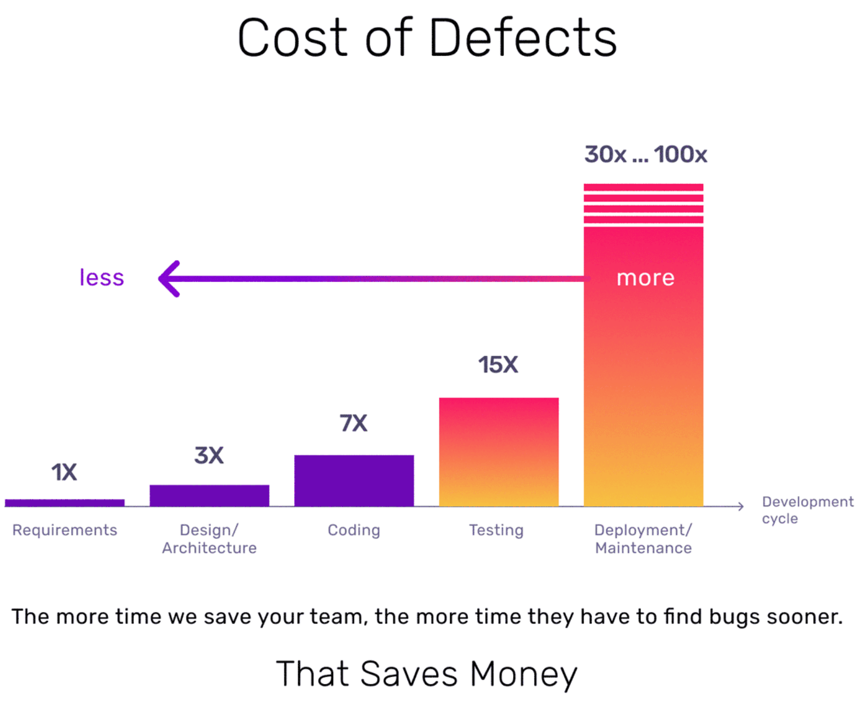


Figure 1: NIST – Cost required to remediate vulnerabilities throughout a SDLC (Cser, 2023)

1.2 Organisation

This report is split up into five sections including related work, tool design, results, evaluation, and conclusion. The related work section discusses how others have tackled the problem of implementing software security and shift left security including the strengths and weaknesses of their approach, and any gaps the approach may have. The tool design section discusses how REPELSEC was developed and its features. The results section provides an overview of the finished deliverable. The evaluation section measures the effectiveness of the REPELSEC tool and evaluates if the requirements were met. The conclusion section summarises what was delivered, the tool's limitations, and future work that would improve the tool.

2 Related Work

2.1 Introduction

In conducting research for this report, industry publications, academic journals, and reputable online resources were explored to find ten popular tools with SAST and/or SCA functionality that could provide valuable insight on how to best approach implementing AST and Shift Left Security. For each tool, its website, documentation, and repository were examined, and a systematic analysis was conducted to find the strengths, weaknesses, and gaps of the tool's approach. Tools with differing and similar approaches to REPELSEC's were studied so that comparative analysis could be performed.

2.2 Tools

2.2.1 Veracode

Veracode (Veracode, 2024) is a closed-source application security cloud platform exclusively used by organisations. It claims to be able to perform SAST and SCA against over 100 different programming languages and frameworks including Java. Unlike REPELSEC, Veracode takes a heavyweight approach to software security meaning the tool performs more complex and lengthy analysis.

Firstly, one way that Veracode has tackled the problem of implementing software security and shift left security is through introducing CI/CD support for various git platforms. One strength of this heavyweight CI/CD approach is its accuracy. The complex nature of Veracode's compilation process and scanning algorithms mean false positives and undetected vulnerabilities are rare. By implementing Veracode into a CI/CD pipeline, security cannot become disregarded within a SDLC. On the other hand, one weakness of this approach is its time-consumption. Veracode's heavyweight approach means that applications must be scanned as whole entities by compiling the source code into binaries. Depending on the size of an application, security analysis can be a long process and can cause concerns for developers. For example, if git commits are being made often, security analysis may obstruct the pipeline resulting in low efficiency CI/CD process.

Secondly, another way that Veracode has tackled the problem of implementing software security and shift left security is by releasing a Visual Studio Code (VSC) extension for more lightweight analysis. One strength of this approach is that a developer can receive visual vulnerability warnings in real time. However, one weakness of this approach is that detailed vulnerability reports cannot be generated. Unlike REPELSEC, this approach also limited portability since the extension is exclusive to the VSC Integrated Development Environment (IDE).

Finally, with Veracode's diversity and support for different languages and frameworks, comes the gap of having less language focused analysis. For example, Veracode appears to focus on only detecting vulnerabilities within MITRE's top 25 Common Weakness Enumerations (CWE) list for each language. However, many common bugs in Java have security implications such as 'CWE-209: Generation of Error Message Containing Sensitive Information' [1] and are not included in this list. By implementing REPELSEC into their testing process, developers can get the benefits of both tools allowing them to conduct complex scans to detect the most severe vulnerabilities using Veracode and make sure other vulnerabilities are not overlooked using REPELSEC.

2.2.2 Semgrep

Semgrep (Semgrep, 2024) is an open-source application security platform with both free and paid tiers used by various types of individual developers and organisations. It claims to support thirty different programming languages including Java. Like REPELSEC, Semgrep has a local CLI tool that focuses on lightweight analysis and shift left ideals. This tool is made up of 'Semgrep Code' which performs SAST against an application, 'Semgrep Supply Chain' which performs SCA against an application, and 'Semgrep secrets' which uses semantic and entropy analysis to look for secrets such as passwords and tokens. One strength of this tool is that there is no need to upload files allowing for fast and local analysis. One weakness of this tool is the number of tests that are run against the code. 600 rules were written by Semgrep's security research team and an additional 2000 tests have been written by their community. If a developer uses too many rules to test, they may be overrun with false positives causing them to become frustrated and desensitised to security issues. Additionally, unlike REPELSEC's testing rules, Semgrep's rules do not strictly look for security issues and may look for other general best code practices. Providing developers with undesirable functionality may lead to a slower tool and unproductive feedback. Like the Veracode platform, the Semgrep tool's approach also has the gap of not being a language focused tool. Rules to detect vulnerabilities outside of the OWASP top 10 or MITRE CWE Top 25 may need to be created by the developer themselves or by the open-source community which may be unreliable.

2.2.3 Spotbugs

Spotbugs (Spotbugs, 2024) is an open-source static analysis tool that looks for potential bugs in Java. Spotbugs takes a different approach compared to many other SAST tools in which the tool is added as a dependency to the Java project through the Maven pom.xml file or Gradle configuration file. Whenever an application is built, feedback is given concerning detected bugs and security issues, tackling the issue of implementing software security and shift left security. Spotbugs also has many plugins for various Java IDEs such as IntelliJ IDEA and Eclipse for real-time feedback. One strength of Spotbugs' approach is its easy integration into Java projects through popular build tools such as Maven and Gradle. This makes it convenient for developers to incorporate this SAST tool into their SDLC without significant overhead. Another strength of this approach is that it is Java focused and can detect bugs that may lead to threats such as resource leaks and thread synchronisation issues. On the other hand, one weakness of Spotbugs' approach is that it

[1] https://cwe.mitre.org/data/definitions/209.html

analyses bytecode rather than the Java source code. Although analysing bytecode has its strengths, it may lack context awareness of where the problem lies in the source code making the tool more complex for developers to use. Additionally, analysing byte code creates a gap in its security coverage. Spotbugs is effective at detecting vulnerabilities related to resource management but may not find more syntax related vulnerabilities such as hard coded credentials. Finally, unlike REPELSEC, Spotbugs does not analyse dependencies or perform SCA against applications.

2.2.4 SonarQube

SonarQube (SonarSource, 2024a) is an open-source platform that performs continuous inspection of code quality and detects vulnerabilities and bugs through static analysis. SonarQube claims to support more than thirty different programming languages and frameworks including Java. For Java, an SCA plugin can also be added to the platform to look for vulnerable dependencies. Like Veracode, SonarQube takes on a heavyweight approach to static analysis which provides comprehensive and accurate analysis of complex vulnerabilities. To tackle the issue of implementing software security and shift left security, SonarQube provides CI/CD integration for many git platforms including GitHub, GitLab, Azure and Bitbucket. One strength of SonarQube's approach is that security is ingrained into the SDLC. Additionally, SonarQube allows for custom security and code quality gates to be configured so that the deployment of a new application version is not released if they do not pass the quality gate tests. On the other hand, one weakness of SonarQube's approach is that it is very resource intensive for organisations compared to most security tools due to the complexity of setting up and configuring this platform. Unlike REPELSEC, one gap of SonarQube's approach is that developers have no way to get fast or real-time feedback for their application as builds must go through a full CI/CD process first.

2.2.5 SonarLint

SonarLint (SonarSource, 2024b) is a closed-source and more lightweight alternative to SonarQube that comes as a plugin for IDEs including Visual Studio, VSC, IntelliJ, and Eclipse. To tackle the issue of implementing software security and shift left security, SonarLint provides real-time feedback for code quality and detected vulnerabilities. One strength of SonarLint's approach is that it uses SonarQube's ruleset so that advanced vulnerabilities can be detected locally enabling a developer to remediate them before code is committed to a CI/CD pipeline. However, one weakness of SonarLint is that it is only supported by four IDEs so if a developer wants to use other popular Java IDEs such as Apache NetBeans or Codenvy, they would not be able to make use of this plugin. One gap of SonarLint is that it has no SCA functionality meaning an additional tool would be needed to perform this analysis and results would not be combined. Additionally, SonarLint's reliance on IDEs and need for a Graphical User Interface (GUI) provides poor testing flexibility whereas REPELSEC can be installed and run from a CLI.

2.2.6 Black Duck

Black Duck (Synopsys, 2024) is a closed-source SCA tool that supports more than twenty different programming languages and package managers. Black Duck takes a comprehensive approach to SCA by offering many types of scan technologies including dependency analysis, binary analysis, code print analysis, and snippet analysis. This enables the scanning of dependency files, artifacts, containers, and firmware to provide a wide coverage of open-source security scanning. To tackle the issue of implementing software security and shift left security, Black Duck has implemented CI/CD integration, build integration with popular Java build tools such as Maven and Gradle, and created plugins for IDEs for more lightweight dependency analysis. One strength of this approach is that developers are given different options to support them in implementing a SSDLC and shift left security. This is important because a developer may find some options more suitable and convenient to implement than others creating a lower chance of security being neglected. However, one weakness of this approach is that it creates a reliance on certain IDEs and pipeline tools which may differ from a developer’s preferred toolset, making them more inefficient. Additionally, Black Duck can be resource intensive due to the number of scanning technologies, making the build process slow. One gap of Black Duck is that it has no SAST functionality meaning an additional tool would be needed to perform this analysis and results would not be combined.

2.2.7 Checkmarx

Checkmarx (Checkmarx, 2024) is closed-source platform that supports many different scanning technologies including SAST and SCA. Checkmarx claims to support over 35 different programming languages and 80 language frameworks. To tackle the issue of implementing software security and shift left security, Checkmarx has implemented CI/CD integration, IDE integration, and promotes their shift-left training. One strength of Checkmarx’s approach is the platform takes training into account which allows for more secure design and implementations from the start of a SDLC. However, one weakness of Checkmarx’s approach is that their platform is resource intensive. Although Checkmarx offers more lightweight analysis with their IDE plugins, this creates a dependency on IDEs which may not be convenient for developers. Additionally, the IDE plugins do not offer SCA functionality creating a gap in which lightweight and local SCA analysis cannot be performed making this plugin potentially inconvenient for developers.

2.2.8 Sonatype Lifecycle

Sonatype’s Lifecycle (Sonatype, 2024) is a closed-source SCA tool that supports over 40 different programming languages and package managers. The Sonatype platform performs dependency analysis, vulnerability detection, license analysis, and policy enforcement for organisations. To tackle the issue of implementing software security and shift left security, Lifecycle integrates into CI/CD pipelines, IDEs as plugins, Java build systems, and allows the tool to be run from a CLI. Like Black Duck, one strength of Lifecycle’s approach is that it gives developers many different options for convenience and even supports command-line integration meaning developers do not have to rely on an IDE. However, one weakness of Lifecycle’s approach is that it can be resource intensive for large codebases. Additionally, Lifecycle is a complex tool that may require significant configuration to meet project requirements making it inconvenient for developers to use. This tool may also be inconvenient because the Sonatype platform has no SAST product that can complement this tool creating a gap. It may be difficult for a developer to find a SAST tool that provides the same shift left benefits and integrations that Lifecycle does.

2.2.9 Snyk

Snyk (Snyk, 2024) is a closed-source platform that performs SAST and SCA against many popular programming languages and frameworks including Java. To tackle the issue of implementing software security and shift left security, Snyk offers solutions including IDE plugins, a CLI alternative, secure programming training for developers, CI/CD integration, and a plugin for Java build tools such as Maven. One strength of Snyk's approach is that it focuses on putting developers first. This is achieved through the number of convenient options provided enabling a developer to conduct security analysis at many stages of development. Unlike many of the previously discussed tools, both SAST and SCA functionality are conducted in parallel with the CLI tool or IDE plugins. The CLI alternative also means developers are not limited to certain IDEs. One weakness of Snyk's approach is that it is not a Java focused security tool, and this may mean fewer vulnerabilities outside of the OWASP Top 25 are tested for within Snyk's SAST functionality.

2.2.10 OWASP DependencyCheck

OWASP DependencyCheck (OWASP, 2024) is an open-source SCA tool developed and maintained by the Open Web Application Security Project (OWASP) community. DependencyCheck supports a wide range of programming languages including Java, Python, and .NET. To tackle the issue of implementing software security and shift left security, OWASP offers their lightweight tool through the command-line or as plugins for the Maven, Ant, and Gradle build tools. One strength of OWASP's approach is their focus on lightweight analysis that can be performed in an IDE or from the command-line. This is often more convenient for a developer than using a platform as it is quick to install and setup within their environment. The integration with build tools also means developers are not limited to a specific IDE that supports a plugin. One weakness of OWASP's approach is the tool's large number of command-line arguments and expansive documentation. Although it offers great customisation and optimisation, a developer not familiar with software security tools may find the number of options overwhelming and confusing, potentially decreasing their efficiency. One gap of OWASP's approach is the limited number of integration options provided for developers. For instance, many of the previously discussed tools offer platforms for vulnerability visualisation, IDE plugins for real-time feedback, and CI/CD integrations with many git platforms allowing for more team-focused software security engineering.

2.3 Summary

To summarise, the explored SAST and SCA tools provided multiple, strong solutions to tackling the issue of implementing software security and shift left security. This was primarily achieved through IDE integration, build tool integration, and CI/CD support. The multiple solutions provided to developers gives them a range of convenient options allowing them to implement a solution that fits their SDLC best. These tools were largely open-source, configurable, and community-driven with support for a range of project requirements. However, these solutions have their limitations.

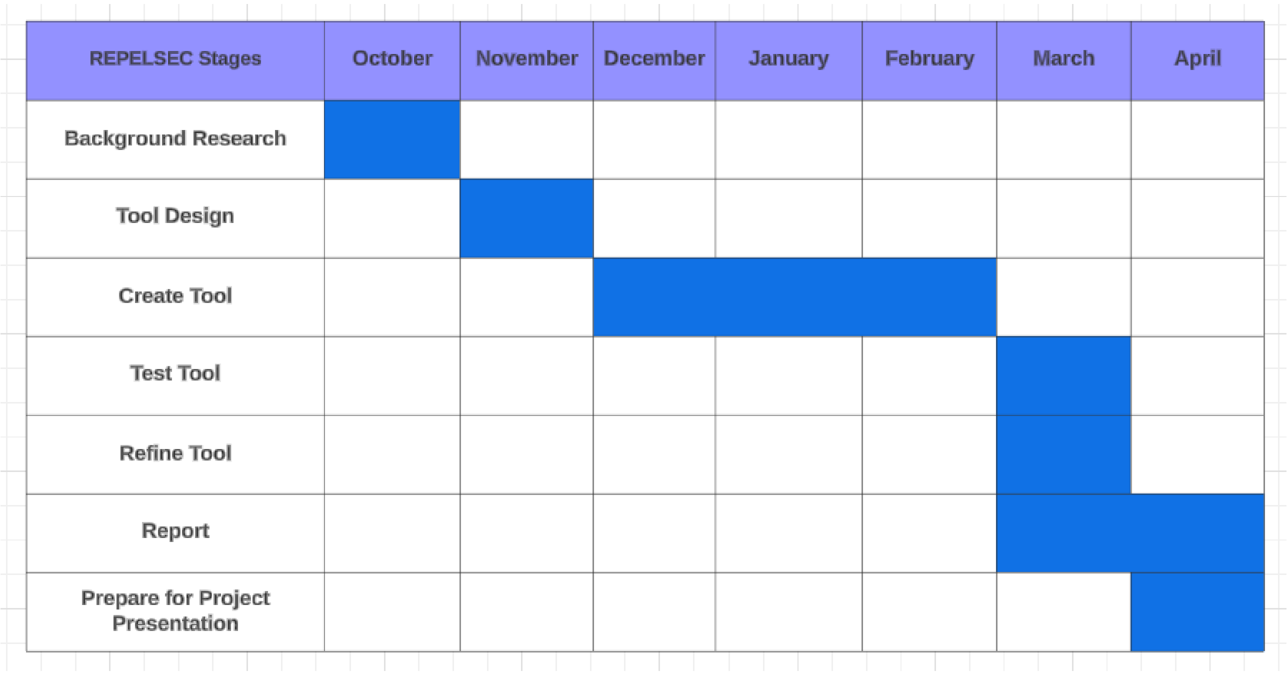
Firstly, IDE integrations from Veracode, Spotbugs, SonarLint, Black Duck, Checkmarx, Sonatype Lifecycle, and Snyk limit developers to certain popular IDEs. Similarly, build tool integrations from Spotbugs, Black Duck, Sonatype Lifecycle, Snyk, and OWASP dependency checker limit developers to certain build tools. These solutions are inconvenient because it means developers would have to switch from the IDE, text editor, or build tool they are familiar with to make use of the integration. Secondly, the lightweight solutions from Veracode, Spotbugs, SonarLint, Black Duck, Checkmarx, Sonatype Lifecycle, and OWASP dependency checker do not provide both SAST and SCA functionality. This creates another inconvenience for developers because it means an additional tool would need to be employed to fill this gap which would result in a more congested and time-consuming security testing process. Finally, all the researched tools except Spotbugs take a diverse approach to AST by supporting a wide range of languages. This can lead to many language-focused vulnerabilities or vulnerabilities outside of the CWE Top 25 being overlooked. As a consequence of these common limitations being identified, REPELSEC was developed with three key objectives including the tool should be flexible, have both SAST and SCA functionality, and should be Java focused.

3 Tool Design

3.1 Project Management

3.1.1 Gantt Chart

Figure 2 illustrates the predicted timeframe and months dedicated for each stage of the project. Stages were allocated with varying timeframes depending on their importance, difficulty, and effort required.



REPELSEC Stages	October	November	December	January	February	March	April
Background Research	🟦						
Tool Design		🟦					
Create Tool			🟦	🟦	🟦		
Test Tool						🟦	
Refine Tool						🟦	
Report						🟦	🟦
Prepare for Project Presentation							🟦

Figure 2: Gantt Chart

3.1.2 Trello

To commence the REPELSEC project, a Trello board was created for project management (See Appendix 3) in which tasks were organised and tracked throughout the SDLC. The board was split up into four stages including ‘To-Do’, ‘In Progress’, ‘To Be Tested’, and ‘Completed’. Each task was represented as a Trello card and assigned a subjective difficulty, effort, and priority rating. Depending on these ratings, an appropriate deadline was assigned. The ‘To-Do’ stage consisted of all the project requirements identified and the priority functional requirements were moved to the ‘In Progress’ stage first. Each Trello card was further split up into more specific actions and tracked using visual cues such as checkboxes.

Trello was chosen over other project management tools because it provides a flexible and adaptive experience. For example, tasks could be added and removed as necessary without causing formatting issues or an overwhelming amount of information to be shown through the interface. Trello also served as a centralised hub where technical resources, background research, and potential bug tracking concerns could be stored without distracting from the main task management stages. This allowed for more streamlined development throughout the SDLC as resources could be quickly accessed and bug concerns could be noted and revisited during the test stage where they would become a focus.

3.1.3 Version Control

The Git tool and GitHub platform were employed to back up the REPELSEC codebase, track changes to the codebase, and release different versions of the tool. These tools were important in enabling project management as commits could be visualised allowing the progress of tasks to be tracked more effectively. These tools were also important in maintaining the integrity of the codebase as frequent commits ensured code was not lost. Additionally, the PyPi repository platform was also employed to enable additional tracking of version releases and to allow REPELSEC users to install and upgrade between releases easily.

3.1.4 Project Risks

Before the development of REPELSEC began, potential risks were identified and recorded so that they could be addressed and managed through project management. Firstly, one technical risk identified was scalability in which the tool may not function correctly when provided with large files to test against. To mitigate this risk, the Trello card associated with this functionality was given an additional action in which further performance testing should be conducted. Secondly, one external risk identified was the reliance on a dependency vulnerability database which is required to conduct SCA. To mitigate this risk, the Trello card was given an additional action in which the appropriate exception handling should be implemented and user feedback should be provided in case the database API is unavailable. Finally, one operational risk identified was the skill gap in security analysis and command-line tool development. To mitigate this risk, extra priority was put on these topics during the background research stage of the project.

3.2 SDLC Methodology

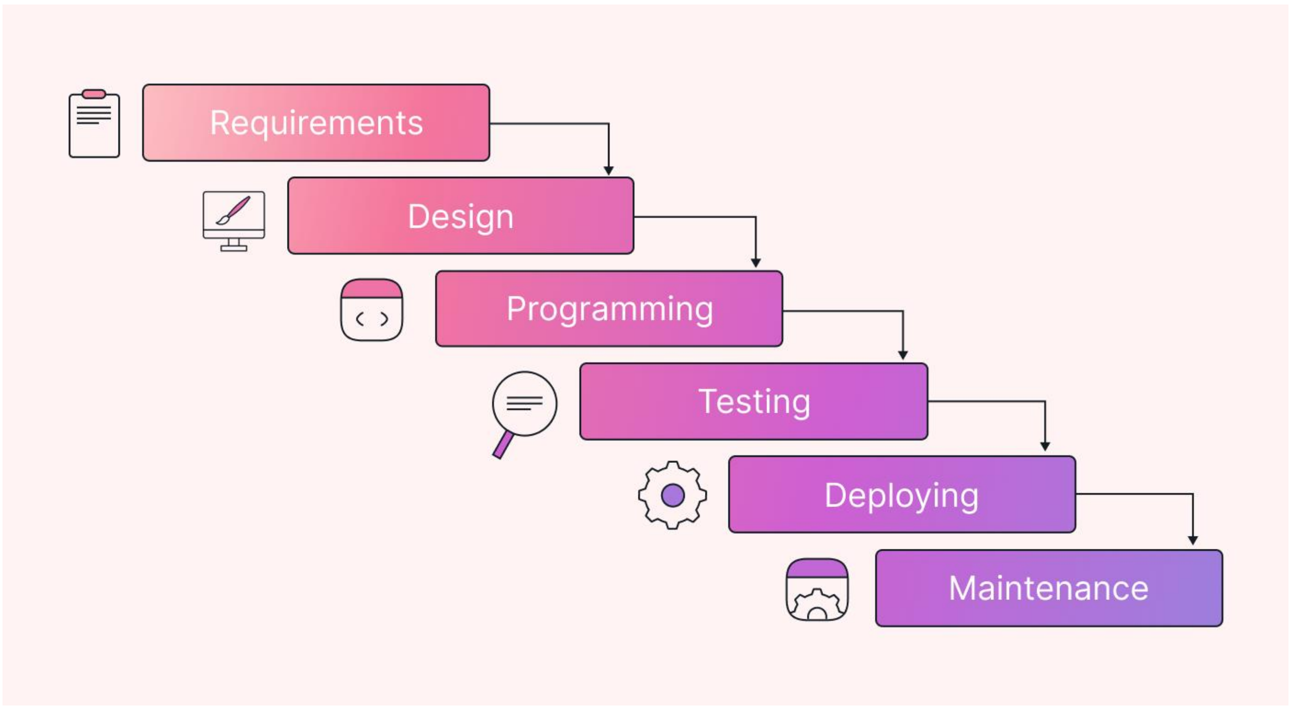


Figure 3: The SDLC Waterfall Method (Motion, 2023)

To develop the REPELSEC tool, the SDLC Waterfall Method was followed. The waterfall method provides a sequential approach to software development through predefined phases. This methodology was chosen over a more agile framework for several reasons. Firstly, a clear and unambiguous set of project requirements were created meaning they would not have to be revised or further specified often. Additionally, there was no external input from stakeholders that would influence the requirements. Secondly, this approach was suitable for a project where the expected workload expectations and deadline was established. Finally, the project allowed for a stable development environment in which tried-and-tested methods such as credible Python dependencies and the NVD could be employed to support the development and meet the requirements within the project timeframe.

3.3 Requirements

When undertaking a large development project, it is important to outline the project requirements to plan for and guide the development process. The identified requirements were based on extensive research of related tools, project supervisor consultations, and analysis of software security best practices. For the REPELSEC tool, functional and non-functional requirements were considered.

3.3.1 Functional Requirements

SAST:

- The tool must support analysis of source code written in the Java programming language.
- The tool must perform scanning for common CWE vulnerabilities associated with Java.
- The tool must report details on any CWE vulnerabilities found including the line number of the source code associated with the vulnerability.

SCA:

- The tool must analyse the third-party dependencies of a Java project and identify vulnerabilities associated with them.
- The tool must integrate with a comprehensive database that keeps up to date with commonly used dependencies, the versioning of each dependency, and published CVEs.
- The tool must report details on any CVE vulnerabilities found including the module affected.

3.3.2 Non-functional Requirements

Performance:

- The tool should perform lightweight analysis in which results are generated quickly and workflows are not interrupted.
- The tool should be able to handle source code and dependency files of varying size.

UI:

- The tool should provide a user-friendly interface accessible to developers who may not be familiar with security tools.
- The tool should provide support to the user by detailing what features are available to them and their function.

Results:

- The tool should provide results in a clear format without overloading the user with unnecessary information.
- The tool could give the user options to export their results to different file formats that suit their needs.
- The tool could give the user options to protect their results.

3.3.3 Use Case Diagram

Figure 4 illustrates the user's potential interactions with the REPELSEC tool and the order in which the functional requirements should exist in the system.

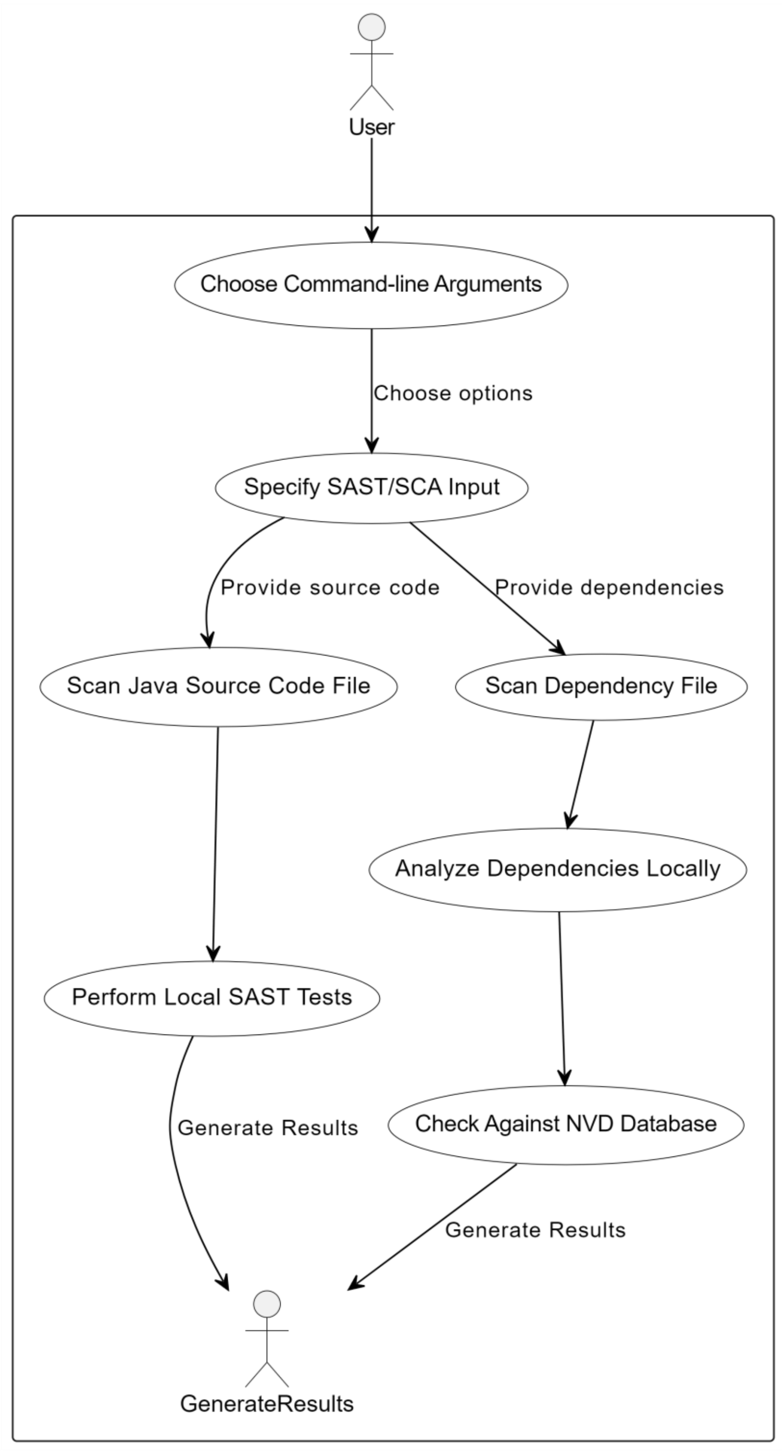


Figure 4: Use Case Diagram

3.4 Design

The design stage of a SDLC aims to translate the requirements into a detailed and technical plan for a project. For the REPELSEC project the architecture, module specification, CLI, PDF Report output, and algorithms were designed before being implemented.

3.4.1 Architecture

Figure 5 outlines the software structure of REPELSEC. It specifies the various abstraction layers of the system and the relationships and key functions that exist within and between them. The tool was designed with a focus on the presentation and applications layers to ensure the tool had a low reliance on external factors such as APIs and databases within the communication and data layers. These factors can be unreliable in terms of performance and availability and bring about additional security implications to consider. As a result, the lightweight nature of REPELSEC could be lost.

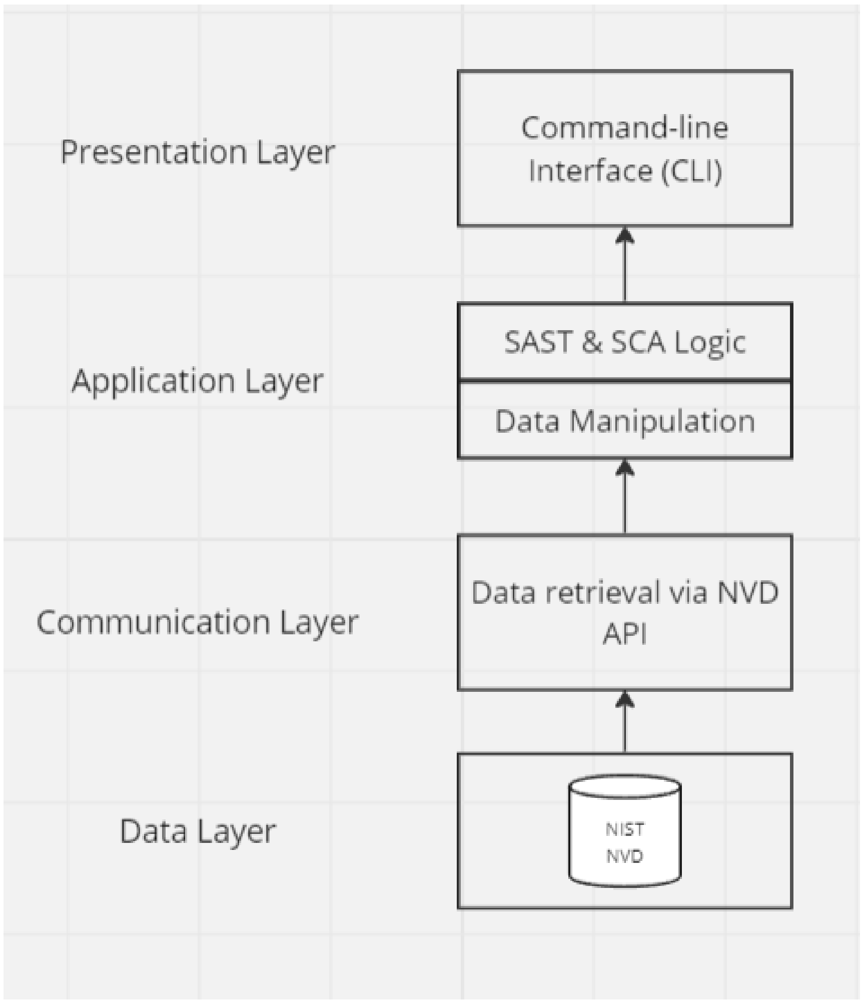


Figure 5: Software Architecture Diagram

3.4.2 Module Specification

Figures 6 illustrates the key modules of REPELSEC and their relationship. The main module was designed to handle many tasks including defining the available command-line arguments, parsing data from source code and dependency files, and printing results. Upon the user providing a source code file to test, the CWE vulnerability module was accessed. This module defined classes for each CWE vulnerability and constructed them with the relevant CWE details. These classes also contained a static method to scan for the vulnerability. On the other hand, upon the user providing a dependency file to test, a local JSON dictionary is accessed. This dictionary translated Java artefact names and versions into their Common Platform Enumeration (CPE) equivalent so that the associated CVE vulnerabilities could be found. Finally, the generate PDF module was accessed if a user enabled the command-line option to create a PDF report. This module defined a visual template and included logic to print vulnerabilities to the file in a structured format.

The module specification was designed like this to create separation between the SAST and SCA functionalities and ensure unnecessary data such as CWE classes were not being loaded into memory when SCA was conducted. This improved performance and contributed to REPELSEC's lightweight design.

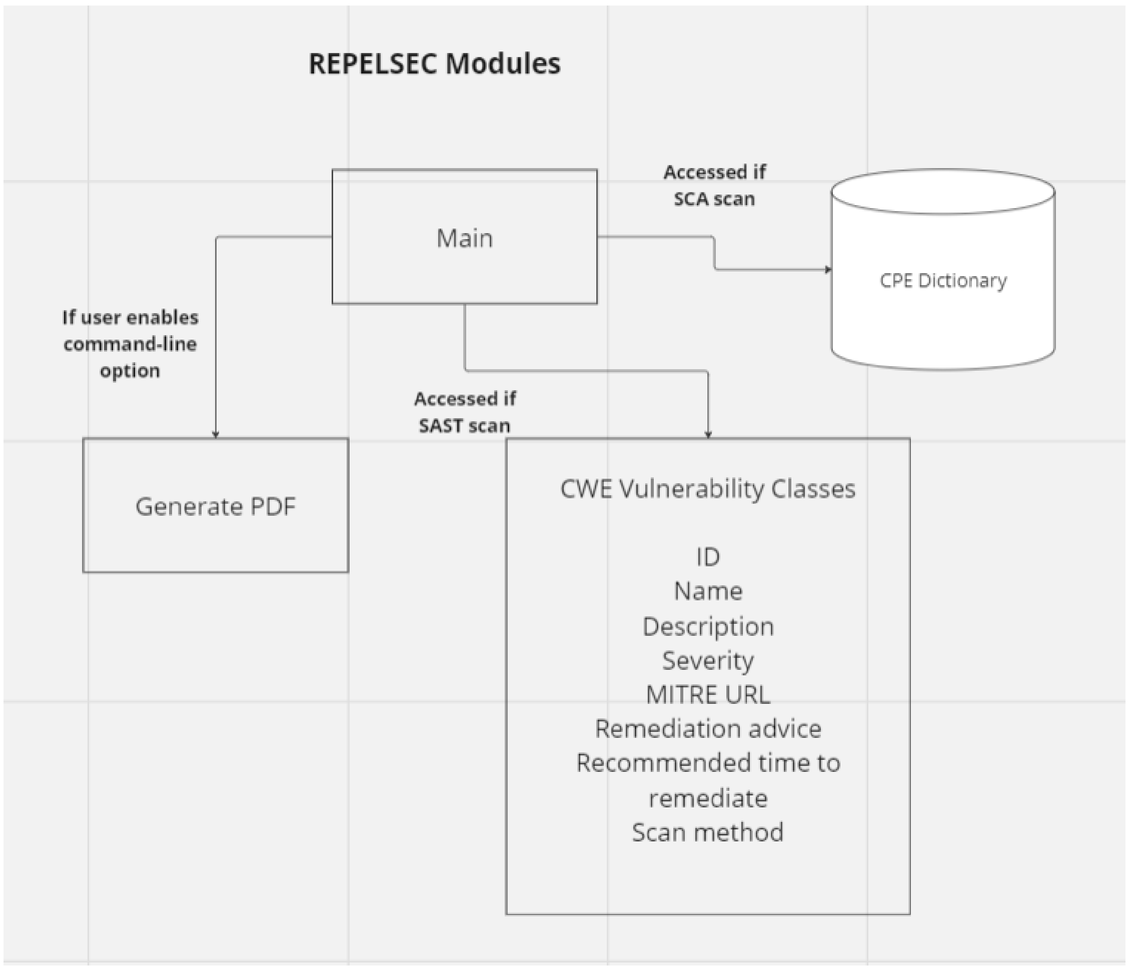


Figure 6: Module Specification Diagram

Figure 7 describes the order of functionalities within the Main module and its interactions with the user.

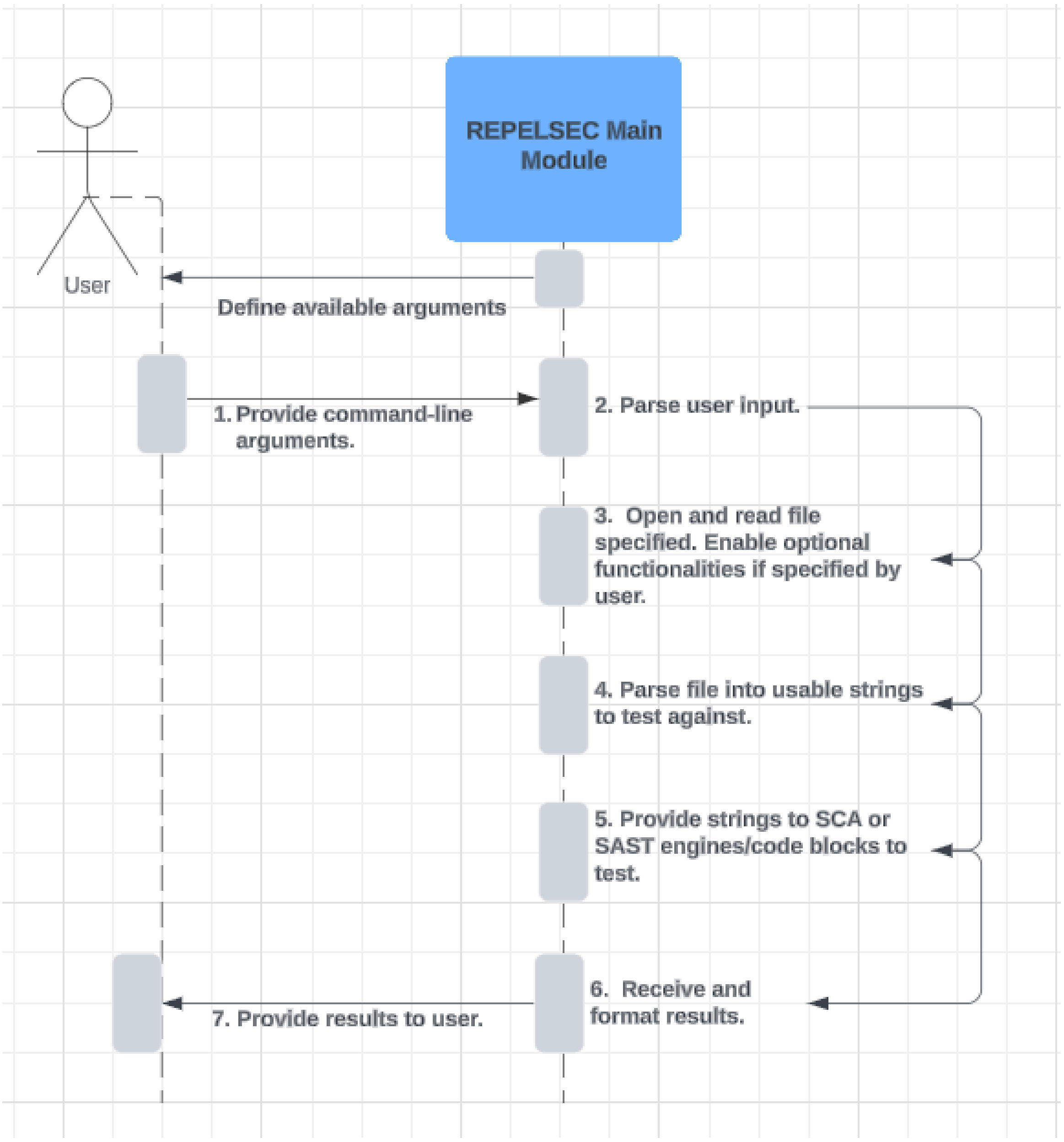

REPELSEC Main Module
User
Define available arguments
1. Provide command-line arguments.
2. Parse user input.
3. Open and read file specified. Enable optional functionalities if specified by user.
4. Parse file into usable strings to test against.
5. Provide strings to SCA or SAST engines/code blocks to test.
6. Receive and format results.
7. Provide results to user.

Figure 7: Main Module Sequence Diagram

3.4.3 Command-line Interface (CLI)

To access the SAST and SCA functionalities, the user must provide the path to the file they would like to test, and the results will be returned through the CLI by default. To achieve the non-functional requirements, optional command-line arguments were designed. The first command-line argument returned a help menu which showed all the command-line options available to the user and described their functionalities. This argument also provided feedback on the positional arguments required and how the command should be structured. The second arguments designed were optional file exports allowing a user to output their results to a text file, a PDF report, or a CSV spreadsheet. These were designed to give users more flexibility in storing, tracking, and visualising their results. The third arguments designed were security commands to allow a user to hide results from their terminal and encrypt their PDF reports. These arguments were implemented to prevent shoulder surfing and to encourage secure sharing of results. The final argument is a convenience option that allows a user to specify the output path for their result files.

Command-line arguments:

- -h / --help – View the help menu.
- -c / --csv - Export results to a CSV file.
- -p / --pdf - Export results to a PDF report.
- -t / --txt - Export results to a TXT file.
- -b / --blank - Stop results from printing to the terminal.
- -e / --password <password> - Encrypt PDF reports.
- -o / --output_path <path> - Specify path to output results to.

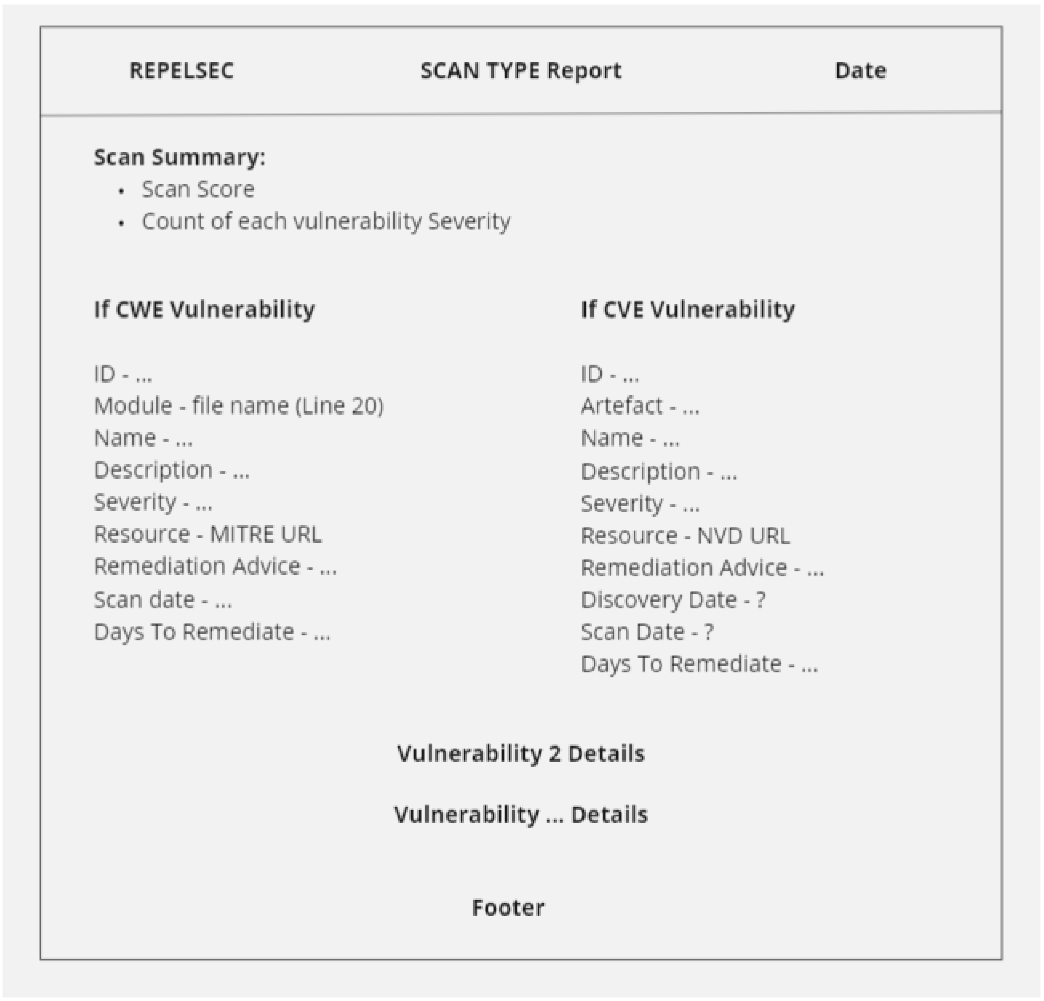
Exception handling and validating user inputs were also considered during the design of the CLI to ensure only valid paths, command-line arguments, file types, and strong passwords could be inputted. To validate these inputs, the Python OS module and regex pattern matching were employed.

The CLI of REPELSEC was carefully designed to meet the requirements and ensure users could access the features of the tool conveniently through the use of command-line options. Additionally, user inputs were validated, and exception handling was implemented to prevent sudden crashes of the tool and give users feedback and an understanding on issues such as the NVD being unavailable.

3.4.4 PDF Report Output

Figure 8 illustrates the design for the PDF Report appearance and the vulnerability details to be included. The report was designed to include a summary of the scan in which a security score out of 100 is given to the file and a count of each vulnerability severity is provided. Developers may find this useful to prioritise which files and vulnerabilities to remediate first. To meet the functional result requirements, the tool provides the relevant vulnerability details, advice on remediating the vulnerability, and the vulnerable module or artefact location.

To not overload a user with information or congest the report, a small selection of details with the most utility was chosen to display to the user. CWE and CVE vulnerability profiles include many details that do not support the user in identifying or remediating a vulnerability. For example, by default, the NVD API returns a dictionary with over thirty different variables for dependency vulnerabilities. Many of these variables have little utility such as scores for outdated Common Vulnerability Scoring System (CVSS) versions like CVSS V2. REPELSEC cuts down on information like this by identifying the latest and most accurate CVSS score and interpreting it as a severity.



REPELSEC **SCAN TYPE Report** **Date**

Scan Summary:
- Scan Score
- Count of each vulnerability Severity

If CWE Vulnerability

ID - ...
Module - file name (Line 20)
Name - ...
Description - ...
Severity - ...
Resource - MITRE URL
Remediation Advice - ...
Scan date - ...
Days To Remediate - ...

If CVE Vulnerability

ID - ...
Artefact - ...
Name - ...
Description - ...
Severity - ...
Resource - NVD URL
Remediation Advice - ...
Discovery Date - ?
Scan Date - ?
Days To Remediate - ...

Vulnerability 2 Details

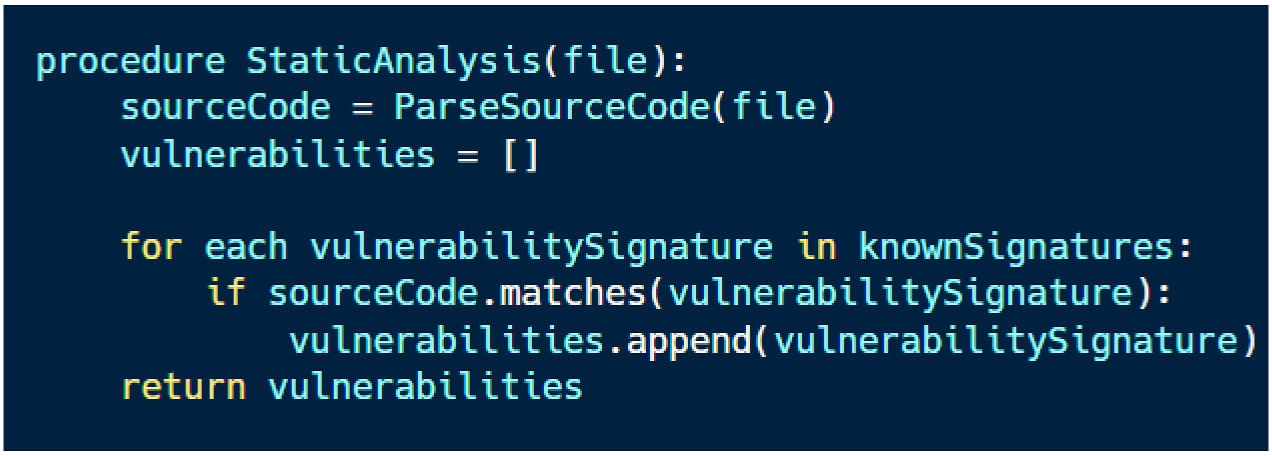
Vulnerability ... Details

Footer

Figure 8: PDF Report Wireframe

3.4.5 Algorithms

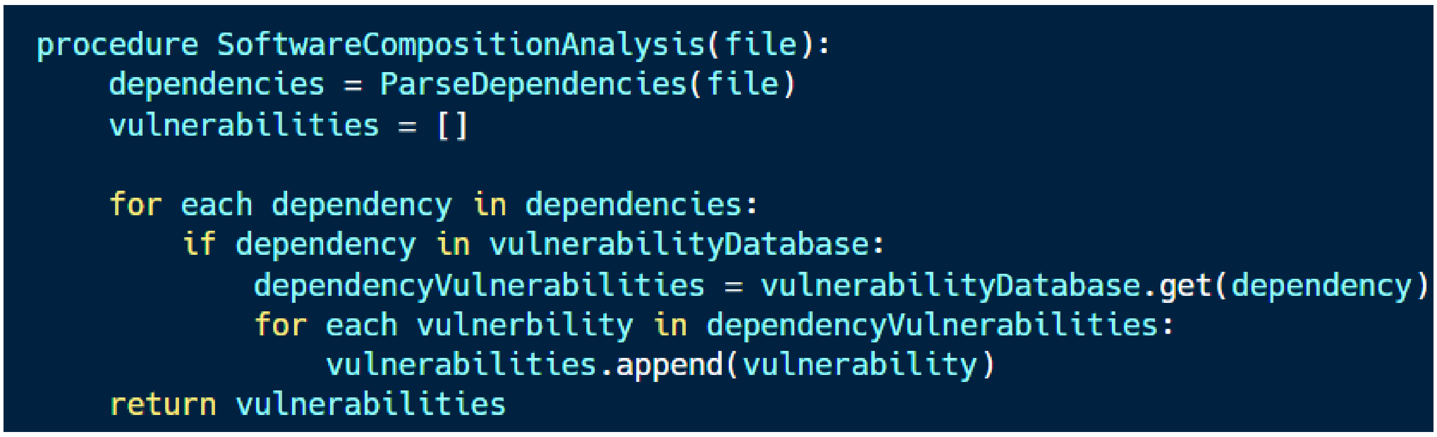
The SAST functionality was implemented in the REPELSEC tool primarily through regex pattern matching. Figure 9 represents the simplified algorithmic description for static analysis in pseudocode. The source code file is parsed into testable strings and then interpreted by each vulnerability scanning function. If the strings match the pattern of the CWE, the vulnerability details are added to a dictionary which tracks all the vulnerabilities identified in the module.



```
procedure StaticAnalysis(file):
    sourceCode = ParseSourceCode(file)
    vulnerabilities = []

    for each vulnerabilitySignature in knownSignatures:
        if sourceCode.matches(vulnerabilitySignature):
            vulnerabilities.append(vulnerabilitySignature)
    return vulnerabilities
```

Figure 9: SAST Simplified Algorithmic Description

The SCA functionality was implemented in REPELSEC by parsing the dependency file and searching for vulnerabilities associated with the dependencies. Figure 10 represents the simplified algorithmic description for SCA in pseudocode. The dependency file is parsed into a list of dependencies and their versions. Each dependency is checked against an external vulnerability database that returns a list of vulnerabilities associated with the dependency. Each of these vulnerabilities are then added to a dictionary which tracks all vulnerabilities present in the dependency file.



```
procedure SoftwareCompositionAnalysis(file):
    dependencies = ParseDependencies(file)
    vulnerabilities = []

    for each dependency in dependencies:
        if dependency in vulnerabilityDatabase:
            dependencyVulnerabilities = vulnerabilityDatabase.get(dependency)
            for each vulnerbility in dependencyVulnerabilities:
                vulnerabilities.append(vulnerability)
    return vulnerabilities
```

Figure 10: SCA Simplified Algorithmic Description

3.5 Implementation (Programming)

The implementation stage of the SDLC aims to translate the design specifications into executable software through programming. The Python programming language was chosen for this task for several reasons. Firstly, Python has flexible string, list, and dictionary manipulation which was suitable when parsing Java source code and dependency files into useful, formatted strings to test against. Secondly, Python packages are quick and easy to install using the Python pip command making it convenient for developers. Thirdly, Python has a wide range of available APIs and libraries to support the development and make sure the promised requirements could be met. Finally, the wide range of forums and technical resources surrounding this language enabled programming challenges to be quickly overcome.

3.6 Testing

3.6.1 Testing Strategy

In the testing stage of the SDLC, REPELSEC underwent rigorous testing to ensure the robustness and reliability of the tool. Testing was important in validating the tool's functionality worked as expected across different environments and in ensuring users will have a similar experience to the developer. A Comprehensive (End-to-End) testing strategy was employed because it covers many aspects of software quality enabling a developer to detect flaws in their design before deployment. The types of tests carried out include unit testing, system testing, and performance testing.

3.6.2 Unit Testing

Unit testing was used to evaluate the individual functions of REPELSEC in isolation and ensure their behaviours worked as expected. The SAST scanning functions were provided with secure and insecure strings to validate that the regex pattern only matched with those that were vulnerable. Additionally, more general functions such as those which calculated a scan security score or returned a recommended remediation timeframe were also tested to validate that they returned the expected results and that the appropriate exception handling was in place to handle unexpected values. 27 unit-tests were created which contained multiple assert statements. These tests all passed (see appendix 4). Figure 11 details the names of the designed unit tests and their purpose.

Unit Test	Description
test_valid_path_function	Assert valid paths return true and invalid paths return false.
test_security_score_function	Assert security score equals certain value depending on severities provided. Additionally, test that security score bounds cannot be breached.
test_get_remediation_days_function	Assert time to remediate equals certain value depending on severities provided.
test_valid_password_function	Assert strong passwords match pattern and weak password do not match pattern.
test_cwe89_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe111_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe209_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe246_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe259_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe321_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe326_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe382_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe395_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe396_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe397_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe481_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe491_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe493_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe500_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe572_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe582_scan	Assert insecure strings match pattern and secure strings do not match pattern.

test_cwe583_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe585_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe586_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe595_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe766_scan	Assert insecure strings match pattern and secure strings do not match pattern.
test_cwe798_scan	Assert insecure strings match pattern and secure strings do not match pattern.

Figure 11: Unit Tests Table

3.6.3 System Testing

System testing was used to evaluate the REPELSEC system as a whole and the functionality of each feature. System testing is important to ensure the end-to-end process works in different scenarios. To achieve this, secure and insecure java source code and dependency files were created or found, and simulations were conducted. Figure 12 details the system tests and their outcomes.

Simulation	Description	Outcome
1	A Java source code file with 6 CWE vulnerabilities was created. The test was conducted to ensure files could be parsed correctly and vulnerabilities could be identified. This test differs from unit testing which provides the string directly to the scanning function.	The file was parsed correctly, and all six vulnerabilities were identified.
2	The pom.xml dependency file was employed from the Javulna (nyilasypeter, 2022) GitHub and tested with REPELSEC. The returned results were manually identified with the NVD website to ensure the API was consistent with the database.	Manual searching of the CPEs on the NVD website found 10 vulnerabilities. This was consistent with the results REPELSEC returned.
3	Files from simulation one and two were tested again with all command-line options enabled so each feature could be observed and validated.	All features worked correctly and upon investigating result outputs, no data inconsistencies were found across different file formats.

Figure 12: System Tests Table

3.6.4 Performance Testing

Performance testing was used to evaluate the tool’s performance under various load conditions (see Appendix 5). The execution time of different file sizes being tested was recorded to ensure the tool remained lightweight and scalable with different loads. Figure 13 details the performance tests and their outcomes.

Test	Description	Outcome
1	A larger Java source code file with 120 lines of code and 24 vulnerabilities was created. Additionally, a file with 45 lines of code and 6 vulnerabilities was created. REPELSEC tested these files and measured the execution time. This test was conducted to ensure the tool could remain lightweight when testing larger files with more vulnerabilities.	The smaller file took 0.003 seconds to execute, and the larger file took 0.03 seconds to execute. This result suggests the tool’s SAST functionality is scalable and there is a negligible difference in performance.
2	The Javulna dependency file was modified and secured. This file has 130 lines of configuration and 3 vulnerabilities. Additionally, the pom.xml dependency file was employed from the Verademo (Veracode, 2023) GitHub. This file has 157 lines of configuration and 13 vulnerabilities. REPELSEC tested these files and measured the execution time. This test was conducted to ensure the requests to the NVD API were not a liability to the performance of the tool.	Both files took 40 seconds to test. This result suggests the number of requests to the NVD API, and amount of data returned and processed, does not have a significant impact on the system.
3	The four files in tests 1 and 2 were tested by REPELSEC with all command-line options enabled. This test was conducted to ensure the features of the tool did not significantly affect performance.	The source code files’ execution times both remained under 0.1 seconds. The Juvulna dependency file took two seconds longer to execute and the Verademo dependency file had no difference. These results suggest the optional features do not affect performance.

Figure 13: Performance Tests Table

3.7 Deployment

In the deployment stage of the SDLC, REPELSEC was readied for distribution. Firstly, the code underwent checks to ensure that all the correct features were enabled and any features in development were commented out or separated from the code to be packaged. Secondly, a readme was created documenting the tool's functionality, features, and how to use it. Thirdly, a Python setup file was created. This file was used to keep track of the tool's versioning, metadata, and the dependencies to be packaged along with application. Fourthly, the Python packaging tools 'setuptools and 'wheel', were employed to generate both source and binary distribution packages of the tool. Once the tool was packaged, it was uploaded to PyPi (Python Package Index) which is the official repository for Python packages. PyPi was chosen because it allows packages to be installed quick and easily using the python 'pip' command, and the repository supports rollbacks of releases. Finally, to ensure supply chain security, the 'twine' tool was employed and configured using strong credentials so that no malicious code could be injected into the packages during an upload.

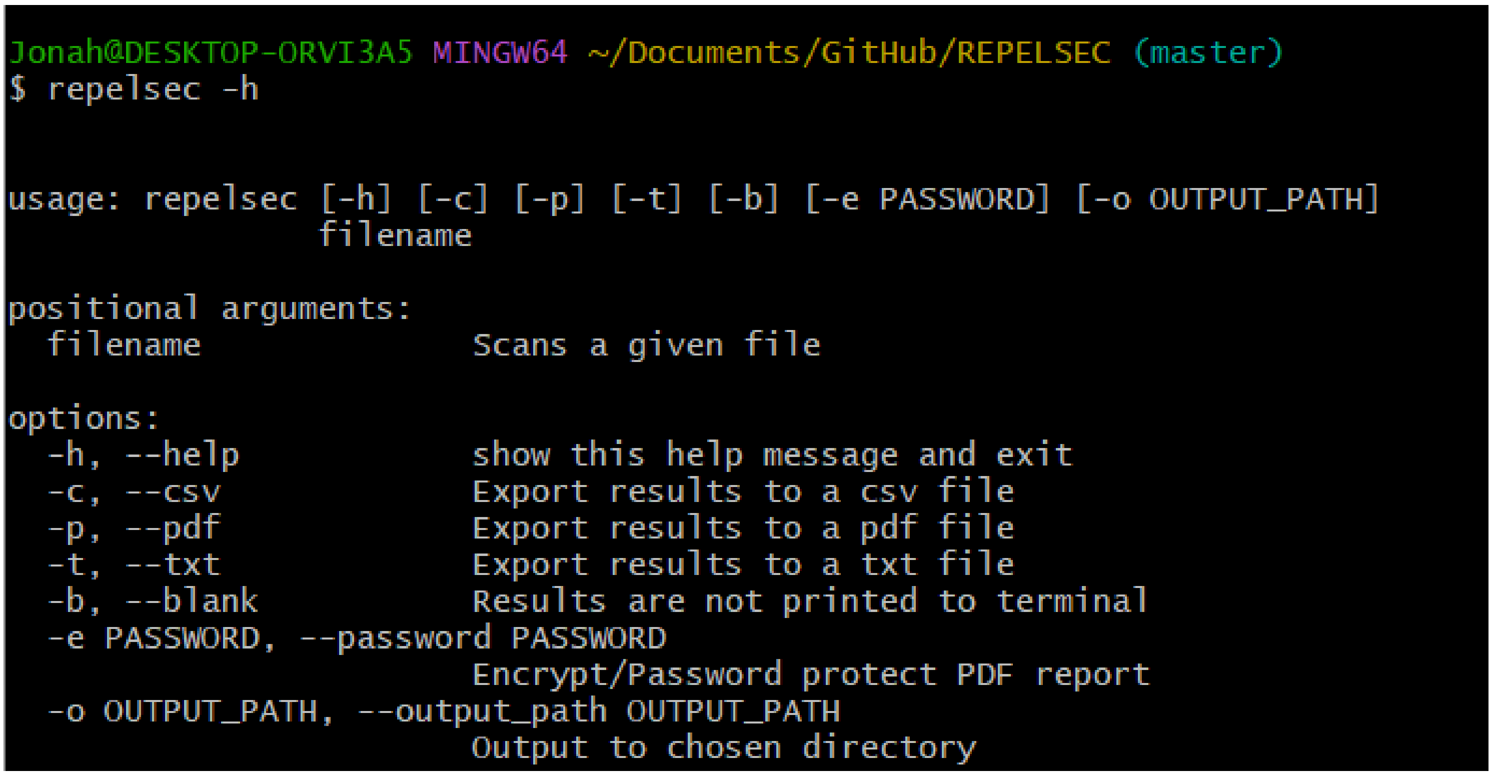
3.8 Maintenance

In the maintenance stage of the SDLC, the focus shifted from development to improving the consistency of the tool and user experience. Following the deployment of the REPELSEC tool, various updates and enhancements were made to the tool to maintain its quality and support. Firstly, identified bugs not found during the testing stage were addressed to further improve upon the tool's accuracy and improve its performance. Additionally, new tests were created to check that similar issues would not arise. Secondly, any new updates to the code or adding and removing of features involved an update to the documentation so that users could always be aware of the tool's functionality in its latest version. Thirdly, dependencies were frequently upgraded to their latest versions so that third-party security and performance issues became less likely to appear. With the upgrade of dependencies, additional testing was also to performed to ensure the tool was still compatible with them. Finally, any gaps in performance were further optimised by replacing inefficient code and removing unnecessary logic.

4 Results

4.1 Features

Users can make use of REPELSEC's SAST and SCA functionalities by inputting the path to a Java source code or dependency file. Additionally, many optional commands are available to users to customise their experience. Users can find and understand these optional features through the documentation or command-line help menu.



```
Jonah@DESKTOP-ORVI3A5 MINGW64 ~/Documents/GitHub/REPELSEC (master)
$ repelsec -h


usage: repelsec [-h] [-c] [-p] [-t] [-b] [-e PASSWORD] [-o OUTPUT_PATH]
                filename

positional arguments:
  filename              Scans a given file

options:
  -h, --help            show this help message and exit
  -c, --csv             Export results to a csv file
  -p, --pdf             Export results to a pdf file
  -t, --txt             Export results to a txt file
  -b, --blank           Results are not printed to terminal
  -e PASSWORD, --password PASSWORD
                        Encrypt/Password protect PDF report
  -o OUTPUT_PATH, --output_path OUTPUT_PATH
                        Output to chosen directory
```

Figure 14: Help Menu

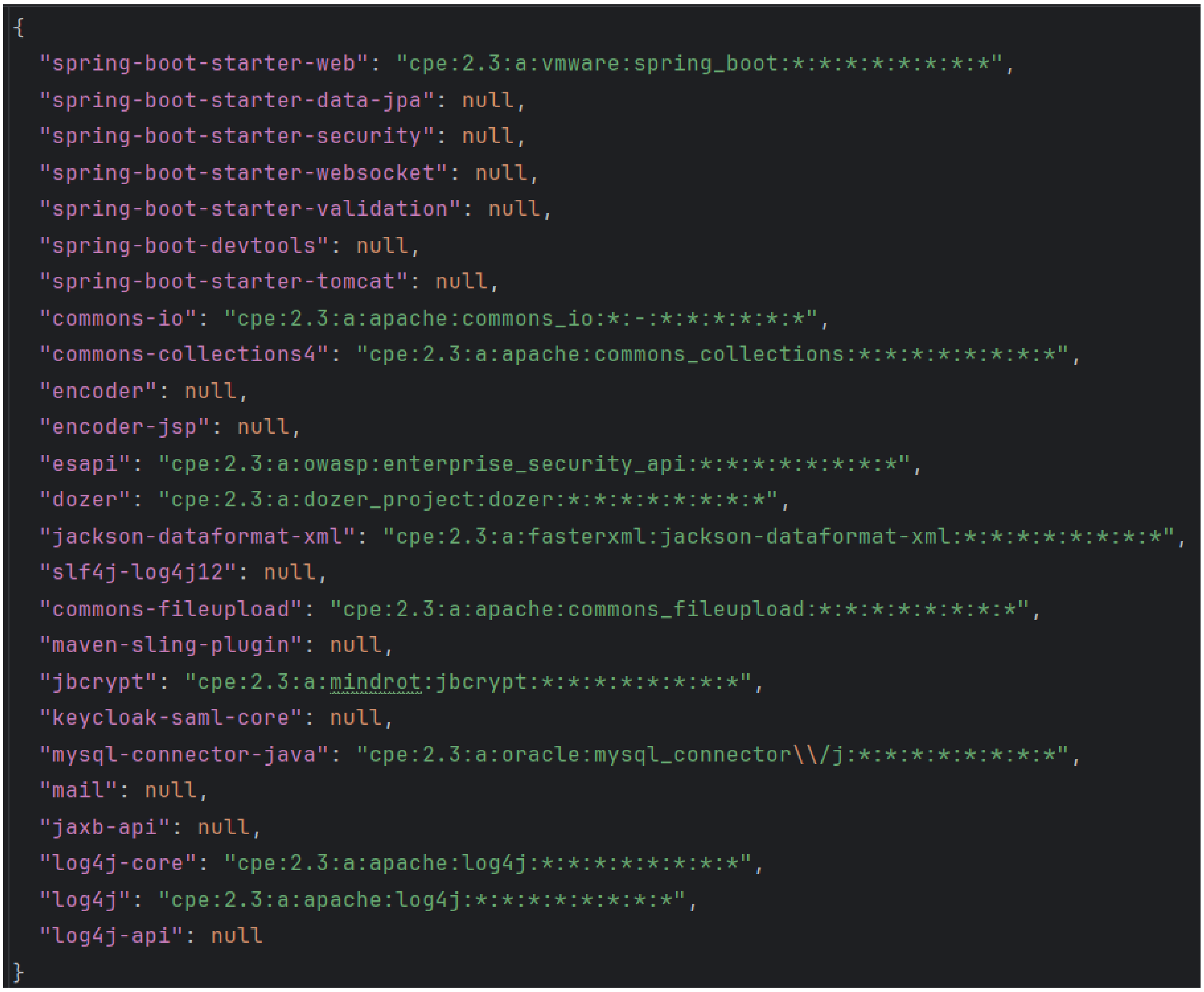
4.2 SAST Functionality

REPELSEC tests Java source code for the below 24 CWE vulnerabilities primarily through regex pattern matching and string queries. Each scanning function underwent unit testing to ensure the regex matched vulnerable strings and not similar, secure strings. The unit tests for the CWE tests all passed (see Appendix 4) suggesting the pattern matching is robust and accurate.

- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- CWE-111: Direct Use of Unsafe JNI
- CWE-209: Generation of Error Message Containing Sensitive Information
- CWE-246: J2EE Bad Practices: Direct Use of Sockets
- CWE-259: Use of Hard-coded Password
- CWE-321: Use of Hard-coded Cryptographic Key
- CWE-326: Inadequate Encryption Strength
- CWE-382: J2EE Bad Practices: Use of System.exit()
- CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference
- CWE-396: Declaration of Catch for Generic Exception
- CWE-397: Declaration of Throws for Generic Exception
- CWE-481: Assigning instead of Comparing
- CWE-491: Public cloneable() Method Without Final ('Object Hijack')
- CWE-493: Critical Public Variable Without Final Modifier
- CWE-500: Public Static Field Not Marked Final
- CWE-572: Call to Thread run() instead of start()
- CWE-582: Array Declared Public, Final, and Static
- CWE-583: finalize() Method Declared Public
- CWE-585: Empty Synchronized Block
- CWE-586: Explicit Call to Finalize()
- CWE-595: Comparison of Object References Instead of Object Contents
- CWE-766: Critical Data Element Declared Public
- CWE-798: Use of Hard-coded Credentials

4.3 SCA Functionality

REPELSEC has a dictionary which translates 26 dependencies to their CPE equivalents or null if the dependency does not have any associated vulnerabilities. These include common libraries such as commons-io and spring-boot-starter-web. The majority of these dependencies were encountered and added to the dictionary while conducting system test 2 and performance test 2 on the Javulna and Verademo dependency files. Additionally, common and infamous dependencies with a history of being vulnerable such as Log4J were researched and added to the dictionary. This functionality represents REPELSEC's capabilities to translate common dependencies into their CPE equivalent. However, there is potential for future work here to automate CPE translations and build a more comprehensive dictionary.

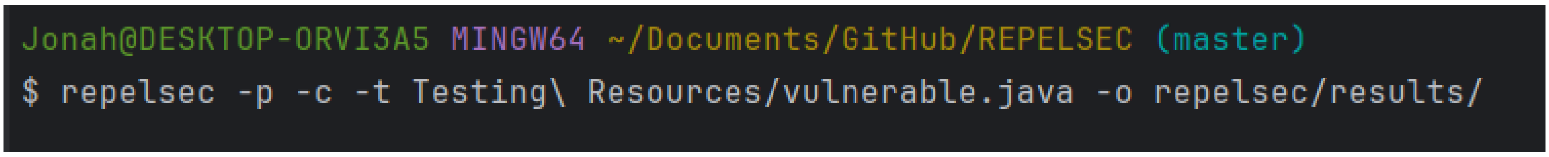


```
{
  "spring-boot-starter-web": "cpe:2.3:a:vmware:spring_boot:*:*:*:*:*:*:*:*",
  "spring-boot-starter-data-jpa": null,
  "spring-boot-starter-security": null,
  "spring-boot-starter-websocket": null,
  "spring-boot-starter-validation": null,
  "spring-boot-devtools": null,
  "spring-boot-starter-tomcat": null,
  "commons-io": "cpe:2.3:a:apache:commons_io:*:-:*:*:*:*:*:*",
  "commons-collections4": "cpe:2.3:a:apache:commons_collections:*:*:*:*:*:*:*:*",
  "encoder": null,
  "encoder-jsp": null,
  "esapi": "cpe:2.3:a:owasp:enterprise_security_api:*:*:*:*:*:*:*:*",
  "dozer": "cpe:2.3:a:dozer_project:dozer:*:*:*:*:*:*:*:*",
  "jackson-dataformat-xml": "cpe:2.3:a:fasterxml:jackson-dataformat-xml:*:*:*:*:*:*:*:*",
  "slf4j-log4j12": null,
  "commons-fileupload": "cpe:2.3:a:apache:commons_fileupload:*:*:*:*:*:*:*:*",
  "maven-sling-plugin": null,
  "jbcrypt": "cpe:2.3:a:mindrot:jbcrypt:*:*:*:*:*:*:*:*",
  "keycloak-saml-core": null,
  "mysql-connector-java": "cpe:2.3:a:oracle:mysql_connector\\/j:*:*:*:*:*:*:*:*",
  "mail": null,
  "jaxb-api": null,
  "log4j-core": "cpe:2.3:a:apache:log4j:*:*:*:*:*:*:*:*",
  "log4j": "cpe:2.3:a:apache:log4j:*:*:*:*:*:*:*:*",
  "log4j-api": null
}
```

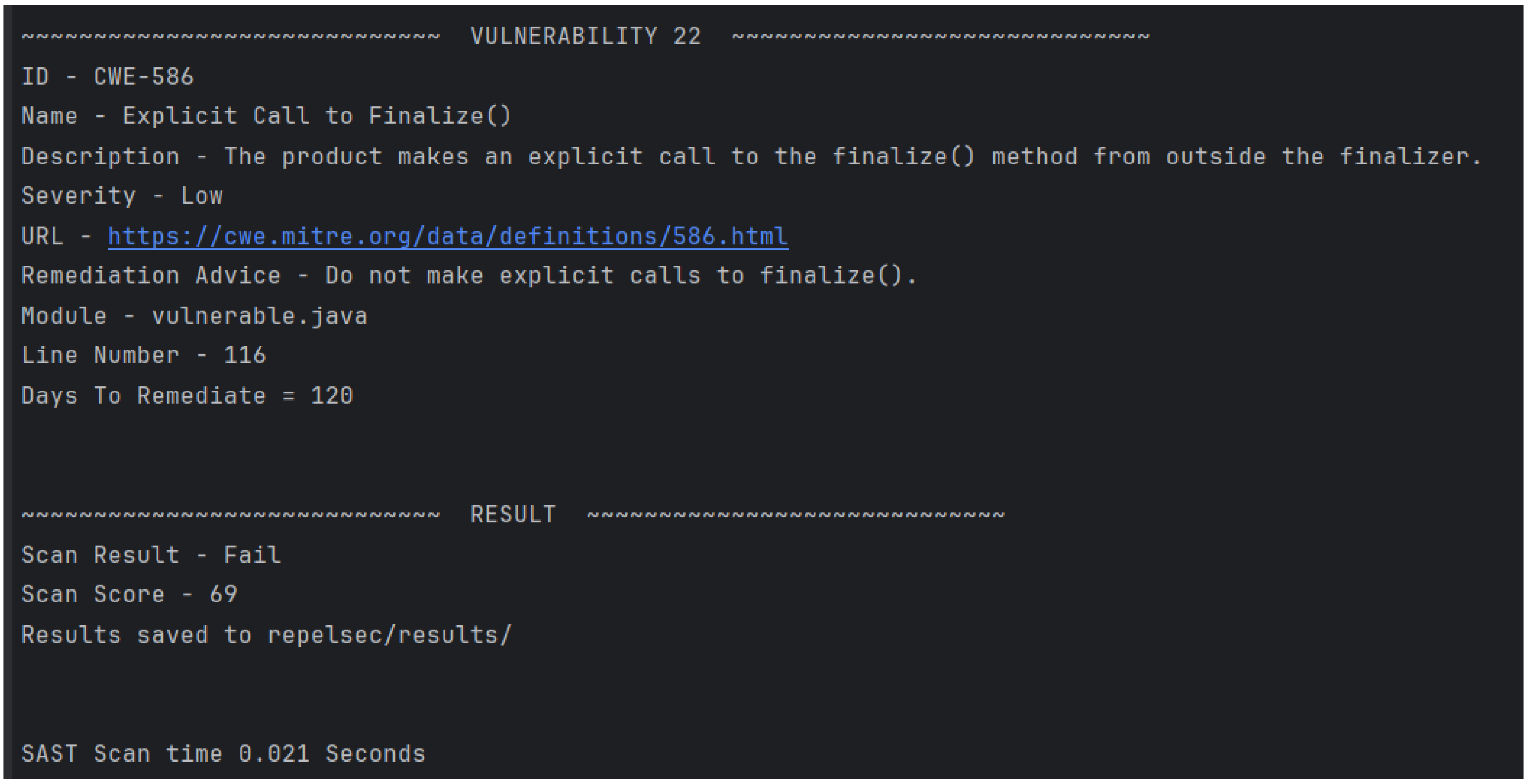
Figure 15: CPE Dictionary

4.4 Result Outputs

By default, results are only printed to the terminal. In addition to the vulnerability details, the terminal provides some further information, including where external files were stored, the scan score, and whether the scan passed or failed. A scan passes if there are no 'High' or 'Critical' severity vulnerabilities found within the scan as further background research found many organisations' risk acceptance allows for "Low" and "Medium" vulnerabilities within their applications.



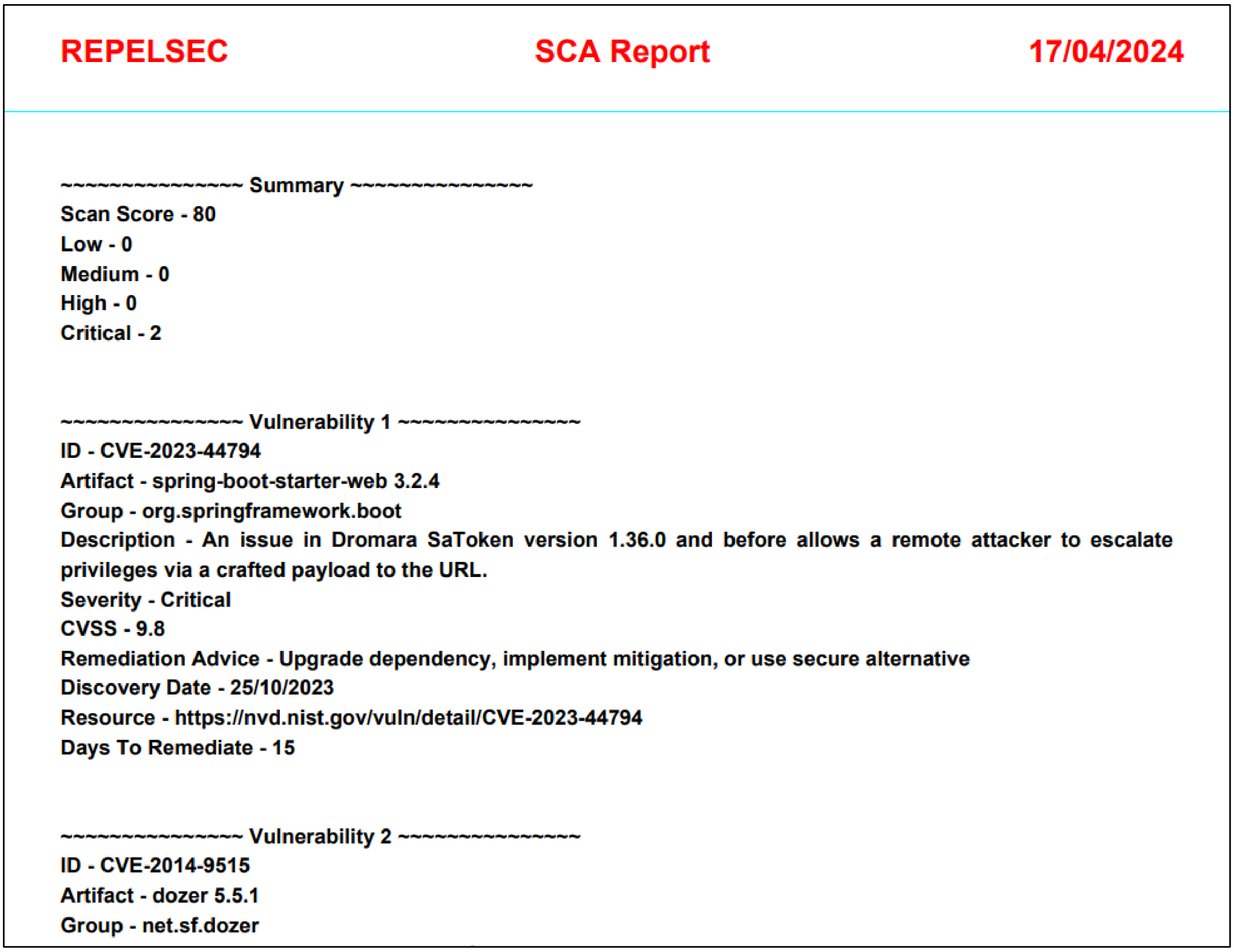
```
Jonah@DESKTOP-ORVI3A5 MINGW64 ~/Documents/GitHub/REPELSEC (master)
$ repelsec -p -c -t Testing\ Resources/vulnerable.java -o repelsec/results/
```



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  VULNERABILITY 22  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
ID - CWE-586
Name - Explicit Call to Finalize()
Description - The product makes an explicit call to the finalize() method from outside the finalizer.
Severity - Low
URL - https://cwe.mitre.org/data/definitions/586.html
Remediation Advice - Do not make explicit calls to finalize().
Module - vulnerable.java
Line Number - 116
Days To Remediate = 120


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  RESULT  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Scan Result - Fail
Scan Score - 69
Results saved to repelsec/results/


SAST Scan time 0.021 Seconds
```

Figure 16: Terminal Output

To export results externally to a file, a user can output their results to a PDF report which includes a summary as seen in Figure 17. Additionally, users can export to a CSV spreadsheet (See Appendix 6) which may be useful in uniting various scan results to form central database and dashboard.



REPELSEC **SCA Report** **17/04/2024**

```
~~~~~~~~~~~~~~~ Summary ~~~~~~~~~~~~~~~
Scan Score - 80
Low - 0
Medium - 0
High - 0
Critical - 2


~~~~~~~~~~~~~~~ Vulnerability 1 ~~~~~~~~~~~~~~~
ID - CVE-2023-44794
Artifact - spring-boot-starter-web 3.2.4
Group - org.springframework.boot
Description - An issue in Dromara SaToken version 1.36.0 and before allows a remote attacker to escalate
privileges via a crafted payload to the URL.
Severity - Critical
CVSS - 9.8
Remediation Advice - Upgrade dependency, implement mitigation, or use secure alternative
Discovery Date - 25/10/2023
Resource - https://nvd.nist.gov/vuln/detail/CVE-2023-44794
Days To Remediate - 15


~~~~~~~~~~~~~~~ Vulnerability 2 ~~~~~~~~~~~~~~~
ID - CVE-2014-9515
Artifact - dozer 5.5.1
Group - net.sf.dozer
```

Figure 17: PDF Report Output

5 Evaluation

5.1 Requirements Tracing

	Requirement	**Was the requirement achieved?**
1	The tool must support analysis of source code written in the Java programming language.	Yes
2	The tool must perform scanning for common CWE vulnerabilities associated with Java.	Yes (To an extent)
3	The tool must report details on any CWE vulnerabilities found including the line number of the source code associated with the vulnerability.	Yes
4	The tool must analyse the third-party dependencies of a Java project and identify vulnerabilities associated with them.	Yes (To an extent)
5	The tool must integrate with a comprehensive database that keeps up to date with commonly used dependencies, the versioning of each dependency, and published CVEs.	Yes
6	The tool must report details on any CVE vulnerabilities found including the module affected.	Yes
7	The tool should perform lightweight analysis in which results are generated quickly and workflows are not interrupted.	Yes
8	The tool should be able to handle source code and dependency files of varying size.	Yes
9	The tool should provide a user-friendly interface accessible to developers who may not be familiar with security tools.	Yes
10	The tool should provide support to the user by detailing what features are available to them and their function.	Yes
11	The tool should provide results in a clear format without overloading the user with unnecessary information.	Yes
12	The tool could give the user options to export their results to different file formats that suit their needs.	Yes
13	The tool could give the user options to protect their results.	Yes

Figure 18: Requirements Tracing Table

The functional SAST requirements were all successfully fulfilled. Requirement 1 was achieved by parsing and testing Java and JSP source code files. This is supported by system test 1 in which these file types were tested. Requirement 2 was achieved to some extent by testing for 24 Java associated CWE vulnerabilities. This is supported by the CWE unit tests and system test 1 in which the SAST functionality was validated. However, this requirement could have been further fulfilled by testing for more severe and complex vulnerabilities within the CWE Top 25 such as CWE-79: Cross-site Scripting[2], CWE-78: OS Command Injection[3], and CWE-73: External Control of File Name or Path.[4] Requirement 3 was achieved by providing the user with the relevant CWE details and the module and line number the vulnerability was found at. This is supported by Figure 16 and system test 1 in which the results of the test were observed.

The functional SCA requirements were all successfully fulfilled. Requirement 4 was achieved to some extent by supporting testing against dependencies within a Maven Pom.xml file. This is supported by system test 2 in which real Maven pom.xml files were tested against. However, this requirement could have been further fulfilled by supporting dependency analysis for other Java build tools such as Gradle. Requirement 5 was achieved by connecting REPELSEC to NIST's NVD which is one of the most reputable databases that tracks dependencies, dependency versions, and their associated CVE vulnerabilities. This is supported by system test 2 and performance test 2 in which the NVD's functionality and performance was validated. Requirement 6 was achieved by providing the user with the relevant CVE details and the associated vulnerable Java artefact. This is supported by Figure 17 which shows the PDF output of a SCA scan.

The non-functional performance requirements were all successfully fulfilled. Requirement 7 was achieved by providing lightweight analysis to the user. This is supported by performance test 1, 2, and 3 in which the performance for all of REPELSEC's features were measured. Requirement 8 was achieved by creating a tool with the ability to perform under various loads. This is supported by performance test 1 and 2 in which files with a differing number of vulnerabilities and file sizes were tested.

The non-functional UI requirements were all successfully fulfilled. Requirement 9 was achieved by implementing a CLI interface with a minimalist feature set to ensure no learning curve was required and to provide a user-friendly experience for those not familiar with security tools. This is supported by Figure 14 and section 3.4.3 which outline the interface available to the user. Requirement 10 was achieved by creating documentation for the tool to list its features and functionality. Additionally, this is supported by Figure 14 which demonstrates a command-line argument to display a help menu. This describes the optional features available to the user and how the command should be structured.

[2] https://cwe.mitre.org/data/definitions/79.html
[3] https://cwe.mitre.org/data/definitions/78.html
[4] https://cwe.mitre.org/data/definitions/73.html

The non-functional results requirements were all successfully fulfilled. Requirement 11 was achieved by only extracting the most important information from the CWE and CVE vulnerability profiles to present to the user. This is supported by Figure 16 and 17 which show the formatted and appropriately spaced results. Requirement 12 was achieved by providing the user with the option to export their results to a PDF file, text file, or CSV spreadsheet file. This is supported by system test 3 and performance test 3 in which the functionality and performance of these features were validated. Requirement 13 was achieved by implementing two features to allow a user to encrypt their PDF files and hide the results from printing in their terminal. This is supported by system test 3 and performance test 3 in which the functionality and performance of these features were validated.

To summarise, all the project requirements were achieved and tested to ensure their validity and performance. Although the functional requirements were implemented sufficiently, there is potential for future work to implement them more extensively.

5.2 Tool Effectiveness & Experiments

Various experiments were conducted to test the effectiveness of the REPELSEC tool.

The first experiment involved finding three popular vulnerable Java projects on GitHub often used for cybersecurity training to test against. These included Javulna, Verademo, and Vulnado (ScaleSec, 2020). The Javulna and Verademo source code and dependency files were used for system and performance testing as detailed in section 3.6. Additional testing was also conducted on a Vulnado source code file named user.java. The outcome of this experiment is that seven vulnerabilities were detected from four types of CWE including CWE-766, CWE-396, CWE-209 and CWE-89. However, one false positive was also returned as CWE-798: Hard-coded credentials being detected. Upon investigating this issue, it was found that the regex in the scanning function for this CWE did not properly account for other use cases for the term ‘Username =’. To tackle this issue, the pattern matching was modified so the term had to be present in a more defined structure to be matched (see Appendix 7). Additionally, more assert statements were added to the unit tests for this CWE and similar CWEs to check for this issue more extensively.

The second experiment involved creating a source code file called vulnerable.java with example code blocks for each vulnerability to test if the tool could effectively find all 24 CWEs. The outcome for this experiment was that the tool detected them all. Additionally, a source code file called secure.java was created with examples of secure code blocks for each vulnerability to test if the tool would return false positives. The outcome of this experiment is that no false positives were returned.

The third experiment involved modifying the dependency file from the first experiment to secure it by upgrading the versions of vulnerable dependencies. This test was conducted to test if the tool could effectively translate artefacts and their versions into the correct CPE equivalents. The outcome of this experiment was that the original vulnerable dependency file returned 10 vulnerabilities and the modified file returned 2 vulnerabilities due to one dependency still being vulnerable on its newest version.

Throughout these experiments, the execution times were noted to look for anomalies and potential failures in the lightweight nature of REPELSEC. The outcome of this is that there were no anomalies for the SAST functionality but there were two instances where the SCA tests took noticeably longer than usual in which execution took two minutes compared to the usual under one minute execution. Upon retesting these files, they took less than a minute which suggests the NVD API can be unreliable in delivering lightweight analysis but is always effective in returning the associated vulnerabilities.

5.3 Tool Comparison: Assessing REPELSEC in Relation to Industry Alternatives

Before the development of the REPELSEC tool, background research into related SAST and SCA security tools was conducted in which various approaches to tackling the issue of implementing software security and shift left security were analysed. Each approach's weaknesses were identified and taken into consideration as objectives when designing and implementing REPELSEC.

The first objective considered during the development of REPELSEC was flexibility for users. Many alternative tools provide IDE integrations as their main lightweight analysis solution. This solution can be unsuitable for some users because it limits them to a certain environment. REPELSEC addresses this issue by being a command-line tool meaning it can be flexibly used across a variety of IDEs and potentially on servers where there is no GUI. However, one strength of IDE integrations is real-time feedback, and some integrations may even allow for auto-correction of code or advanced machine learning analysis of a developer's programming. As a result, it could be argued that REPELSEC takes a more flexible approach but loses some utility when compared to IDE integrations.

The second objective considered during the development of REPELSEC was convenience. Many lightweight solutions only provide SAST or SCA functionality. This can be inconvenient for developers as it means another tool would have to be employed. This can result in an increasingly more complex CI/CD pipeline and SDLC process. REPELSEC addresses this issue by including both functionalities but not forcing both to be conducted at the same time to ensure analysis remains lightweight. However, one strength of focusing on one functionality is that more resources can be dedicated to one functionality enabling it to become more specialised and accurate. Additionally, some developers may only want to employ one functionality and hence install a more specialised tool.

The third objective considered during the development of REPELSEC was language specialisation. Research into related tools found many tools took a diverse approach to analysis in which over thirty programming languages and frameworks were commonly supported. This can result in less language focused analysis where vulnerabilities outside of MITRE's CWE Top 25 or more Java associated vulnerabilities such as those related to modifiers are neglected. REPELSEC addresses this issue by being Java-focused and attempting to test for as many CWEs as possible no matter the likelihood or severity of the CWE. However, one strength of a diverse approach is that it can be employed by more users and organisations in which many different programming languages are used. Additionally, it could be argued the focus on the OWASP Top 25 is more beneficial as preventing the most dangerous and likely vulnerabilities is important when implementing software security.

6 Conclusion

6.1 Promises and Accomplishments

REPELSEC represents a significant milestone in enhancing the software security of Java projects and enabling developers to incorporate the concepts of Shift Left Security and secure software engineering. The tool’s greatest accomplishment is that it can seamlessly integrate into a developer’s workflow and the SDLC. The tool offers valuable capabilities for identifying vulnerabilities in Java source code and dependencies while remaining lightweight, efficient, and flexible. Through thorough research on related tools, a meticulous SDLC process, and an evaluation of the tool’s effectiveness, the promise of creating a lightweight command-line SAST and SCA tool was delivered.

6.2 Difficulties, Limitations, and Unfulfilled Ambitions

Throughout the development of the REPELSEC, difficulties were encountered which led to unfulfilled ambitions and limitations for the tool. Although these can be addressed through future work to the tool, it is important to reflect on the challenges faced so that they can be considered when designing and implementing future features. One ambition left unfilled was the implementation of the SAST functionality. Although the requirement was achieved to some extent, the tool cannot detect every CWE vulnerability associated with the Java language. This was due to the difficulty of implementing the various types of static analysis techniques available. The REPELSEC tool was designed with pattern matching and single file analysis as the focus. This meant tests for vulnerabilities such as reflected XSS were hard to implement because this vulnerability often spans across multiple files and is not purely a syntax issue. However, the REPELSEC tool does serves its purpose in detecting common security bugs during the coding stage of a SDLC and when combined with a heavyweight SAST/SCA cloud platform such as Veracode, it is very beneficial to a user in ensuring software security.

6.3 Audience and Profitability

The REPELSEC tool caters to a diverse audience of IT users including software engineers, DevSecOps teams, and application security engineers. One problem these users face is that security tools often limit them to a specific environment which may not be ideal. For example, although IDE plugins offer many benefits such as real-time feedback, some developers may be more efficient when working with a more familiar IDE that does not support the plugin or a lightweight text editor. REPELSEC is a solution to this issue as it can be used anywhere through the command-line. The simple CLI and minimal feature set of the tool may also appeal to users not familiar with application security tools as it does not require a steep learning curve. Finally, the tool offers both SAST and SCA functionality making the testing process more streamlined and less complex which will especially appeal to software engineers whose time is focused on implementing new features rather than security fixes.

6.4 Important Features and Implementations

Key features of the REPELSEC tool include its ability to perform both SAST and SCA from the command-line, allow for customisable result outputs with protection features, and produce professional PDF reports that provide users with important vulnerability details and remediation guidance. The tool was implemented with Python which allowed for Java files to be parsed flexibly and efficiently. Additionally, the use of reputable packaging tools and the PyPi deployment platform provides users with a simple installation process making the tool convenient to add to a user's workflow.

6.5 Future Work

To further refine the REPELSEC tool, there are several opportunities that would provide a better user experience and improve the utility of the tool. Firstly, to improve the SAST functionality, a wider range of support for vulnerabilities could be implemented including 'CWE-79: Cross-site Scripting' and 'CWE-78: OS Command Injection'. To achieve this, more advanced static analysis techniques would need to be incorporated such as data flow, control flow, and taint analysis. Secondly, to ensure the longevity of the SCA functionality, an API could be developed that generates an updated CPE dictionary so that the tool always has the most up to date CPE equivalents for Java dependencies. Finally, the remediation advice could be improved by providing the user with suggested secure code alterations which may give them a clearer idea of how to secure their code practically and not just in theory.

References

Application, S. -. I. V. J., 2020. *Vulnado -.* [Online]
Available at: https://github.com/ScaleSec/vulnado
[Accessed 20 April 2024].

Checkmarx, 2024. *Checkmarx: Application Security Testing Tool.* [Online]
Available at: https://checkmarx.com/
[Accessed 31 March 2024].

Cser, T., 2023. *The Cost of Finding Bugs Later in the SDLC.* [Online]
Available at: https://www.functionize.com/blog/the-cost-of-finding-bugs-later-in-the-sdlc
[Accessed 2 April 2024].

Khan, M. U. A. & Zulkernine, M., 2009. On selecting appropriate development processes and requirements engineering methods for secure software.. In: *2009 33rd Annual IEEE International Computer Software and Applications Conference.* s.l.:IEEE, pp. 353-358.

Motion, 2023. *The Power of the Waterfall Model.* [Online]
Available at: https://www.usemotion.com/blog/waterfall-model#when-the-waterfall-is-king
[Accessed 2 April 2024].

nyilasypeter, 2022. *Javulna - A deliberately vulnerable java app for educational purposes.* [Online]
Available at: https://github.com/edu-secmachine/javulna
[Accessed 20 April 2024].

OWASP, 2024. *OWASP Dependency-Check.* [Online]
Available at: https://owasp.org/www-project-dependency-check/
[Accessed 1 April 2024].

Pitchford, M., 2021. The ‘Shift Left’Principle.. In: s.l.:s.n., pp. 18-21.

Semgrep, 2024. *Semgrep — Find bugs and enforce code standards.* [Online]
Available at: https://semgrep.dev/
[Accessed 29 March 2024].

SonarSource, 2024a. *Code Quality, Security & Static Analysis Tool with SonarQube.* [Online]
Available at: https://www.sonarsource.com/products/sonarqube/
[Accessed 30 March 2024].

SonarSource, 2024. *Linter IDE Tool & Real-Time Software for Code.* [Online]
Available at: https://www.sonarsource.com/products/sonarlint/
[Accessed 30 March 2024].

Sonatype, 2024. *Sonatype Lifecycle - Application Security Testing.* [Online]
Available at: https://www.sonatype.com/products/open-source-security-dependency-management
[Accessed 31 March 2024].

SpotBugs, 2024. *SpotBugs.* [Online]
Available at: https://spotbugs.github.io/
[Accessed 30 March 2024].

Synk, 2024. *Snyk | Developer security | Develop fast. Stay secure. | Snyk.* [Online]
Available at: https://snyk.io/
[Accessed 1 April 2024].

Synopsys, 2024. *Black Duck Software Composition Analysis (SCA).* [Online]
Available at: https://www.synopsys.com/software-integrity/software-composition-analysis-tools/black-duck-sca.html
[Accessed 31 March 2024].

Veracode, 2023. *verademo - A deliberately insecure Java web application.* [Online]
Available at: https://github.com/veracode/verademo
[Accessed 20 April 2024].

Veracode, 2024. *Veracode - Leaders in Security | Learn How To Prioritize Risks.* [Online]
Available at: https://www.veracode.com/
[Accessed 29 March 2024].

7 Appendices

7.1 Appendix 1: Code

Sources:

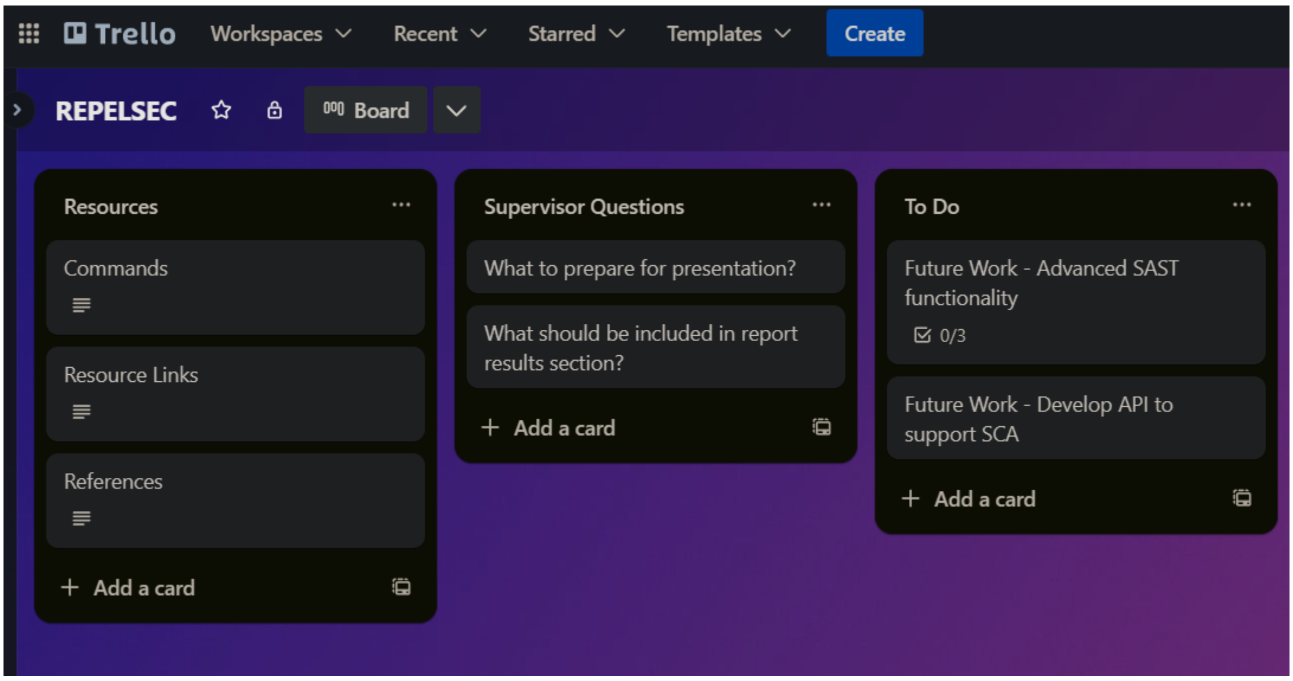
- GitHub - https://github.com/xJonah/REPELSEC.
- PyPi - https://pypi.org/project/repelsec.

7.2 Appendix 2: Supervisor Meetings

Topics discussed with supervisor:

- Project requirements.
- Technical advice concerning static analysis and Python.
- Report structure.
- Task management and deadline planning.

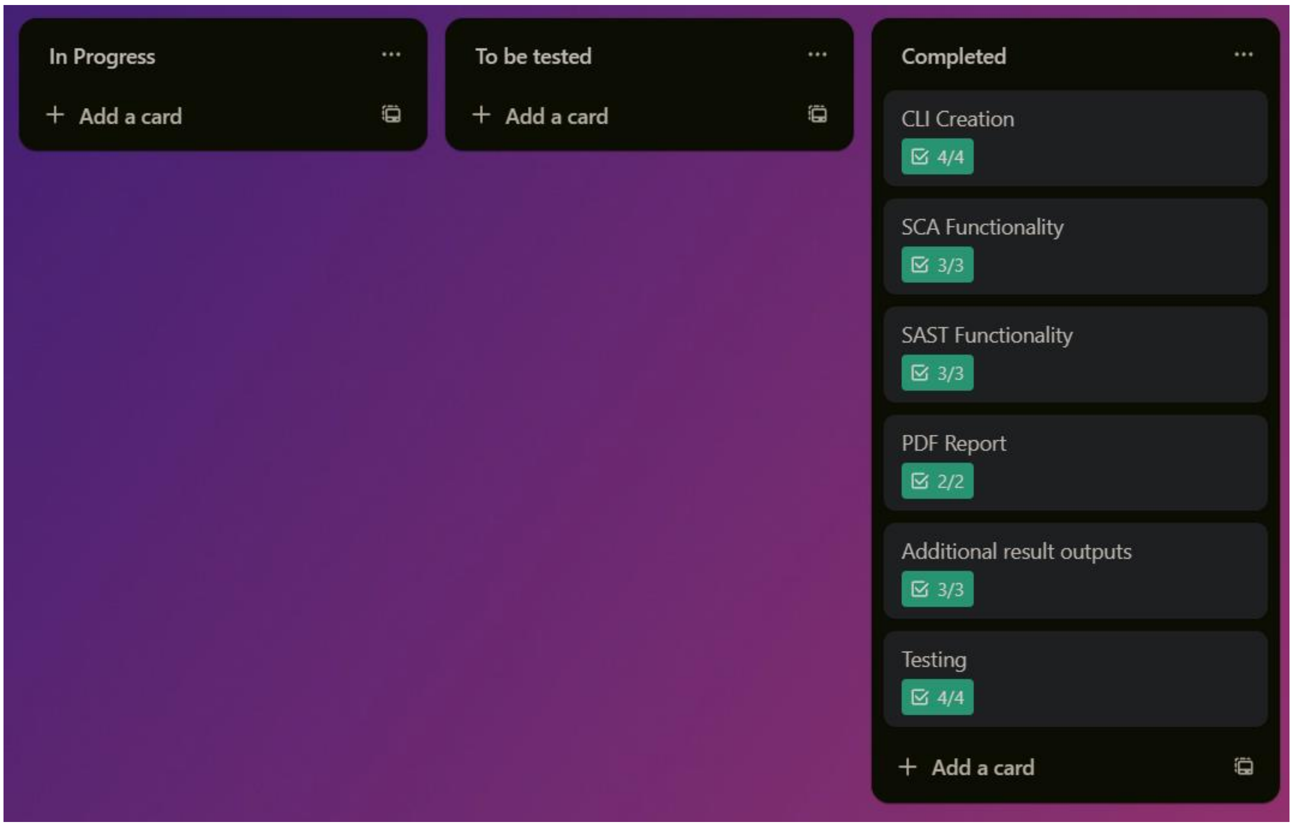
7.3 Appendix 3: Trello Board



Resources
- Commands
- Resource Links
- References
- + Add a card

Supervisor Questions
- What to prepare for presentation?
- What should be included in report results section?
- + Add a card

To Do
- Future Work - Advanced SAST functionality ☑ 0/3
- Future Work - Develop API to support SCA
- + Add a card

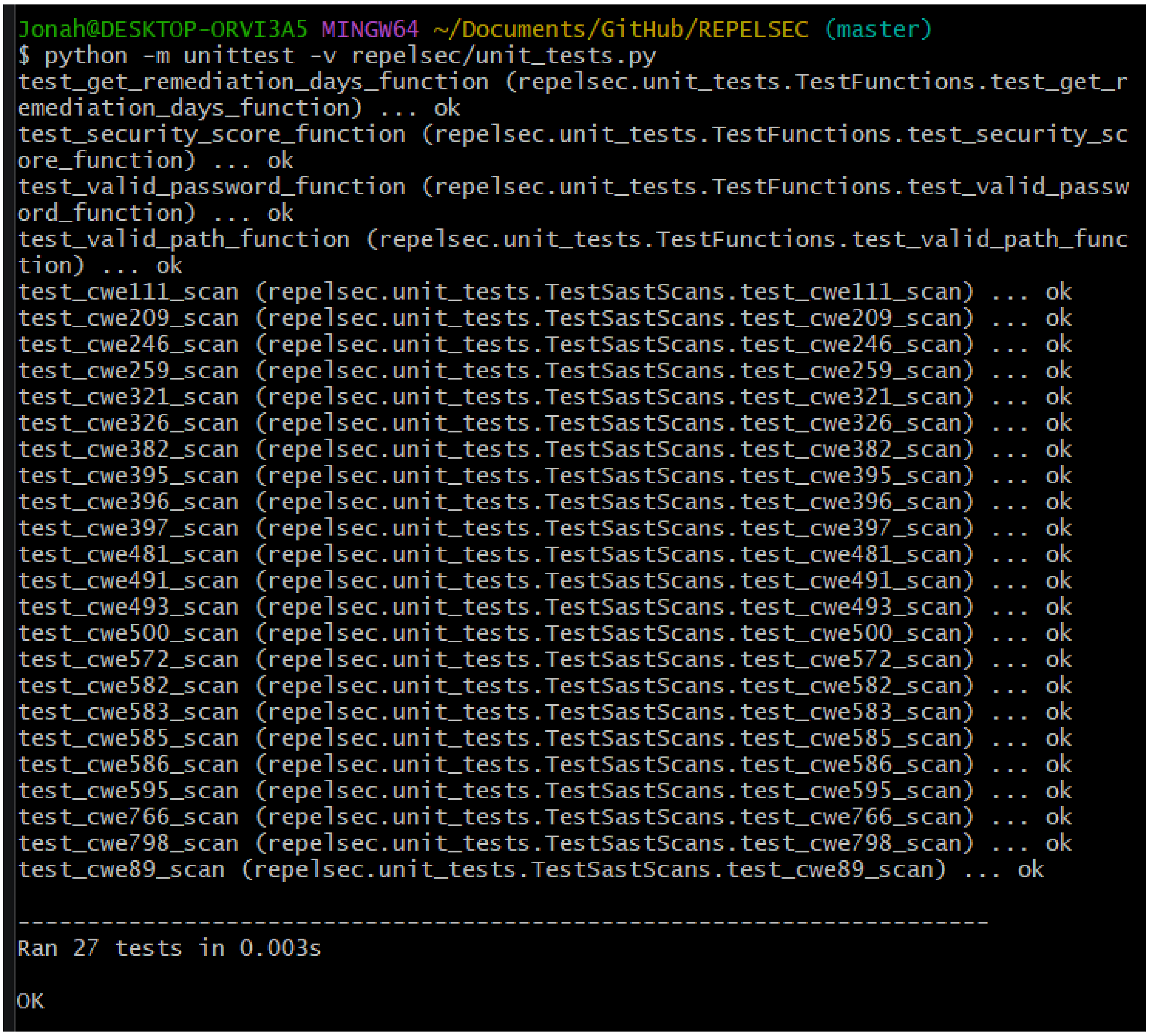


In Progress
- + Add a card

To be tested
- + Add a card

Completed
- CLI Creation ☑ 4/4
- SCA Functionality ☑ 3/3
- SAST Functionality ☑ 3/3
- PDF Report ☑ 2/2
- Additional result outputs ☑ 3/3
- Testing ☑ 4/4
- + Add a card

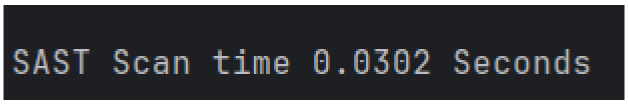
7.4 Appendix 4: Unit Tests



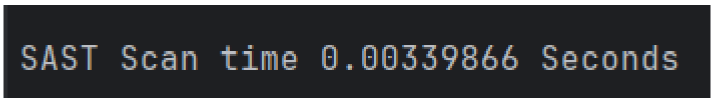
```
Jonah@DESKTOP-ORVI3A5 MINGW64 ~/Documents/GitHub/REPELSEC (master)
$ python -m unittest -v repelsec/unit_tests.py
test_get_remediation_days_function (repelsec.unit_tests.TestFunctions.test_get_r
emediation_days_function) ... ok
test_security_score_function (repelsec.unit_tests.TestFunctions.test_security_sc
ore_function) ... ok
test_valid_password_function (repelsec.unit_tests.TestFunctions.test_valid_passw
ord_function) ... ok
test_valid_path_function (repelsec.unit_tests.TestFunctions.test_valid_path_func
tion) ... ok
test_cwe111_scan (repelsec.unit_tests.TestSastScans.test_cwe111_scan) ... ok
test_cwe209_scan (repelsec.unit_tests.TestSastScans.test_cwe209_scan) ... ok
test_cwe246_scan (repelsec.unit_tests.TestSastScans.test_cwe246_scan) ... ok
test_cwe259_scan (repelsec.unit_tests.TestSastScans.test_cwe259_scan) ... ok
test_cwe321_scan (repelsec.unit_tests.TestSastScans.test_cwe321_scan) ... ok
test_cwe326_scan (repelsec.unit_tests.TestSastScans.test_cwe326_scan) ... ok
test_cwe382_scan (repelsec.unit_tests.TestSastScans.test_cwe382_scan) ... ok
test_cwe395_scan (repelsec.unit_tests.TestSastScans.test_cwe395_scan) ... ok
test_cwe396_scan (repelsec.unit_tests.TestSastScans.test_cwe396_scan) ... ok
test_cwe397_scan (repelsec.unit_tests.TestSastScans.test_cwe397_scan) ... ok
test_cwe481_scan (repelsec.unit_tests.TestSastScans.test_cwe481_scan) ... ok
test_cwe491_scan (repelsec.unit_tests.TestSastScans.test_cwe491_scan) ... ok
test_cwe493_scan (repelsec.unit_tests.TestSastScans.test_cwe493_scan) ... ok
test_cwe500_scan (repelsec.unit_tests.TestSastScans.test_cwe500_scan) ... ok
test_cwe572_scan (repelsec.unit_tests.TestSastScans.test_cwe572_scan) ... ok
test_cwe582_scan (repelsec.unit_tests.TestSastScans.test_cwe582_scan) ... ok
test_cwe583_scan (repelsec.unit_tests.TestSastScans.test_cwe583_scan) ... ok
test_cwe585_scan (repelsec.unit_tests.TestSastScans.test_cwe585_scan) ... ok
test_cwe586_scan (repelsec.unit_tests.TestSastScans.test_cwe586_scan) ... ok
test_cwe595_scan (repelsec.unit_tests.TestSastScans.test_cwe595_scan) ... ok
test_cwe766_scan (repelsec.unit_tests.TestSastScans.test_cwe766_scan) ... ok
test_cwe798_scan (repelsec.unit_tests.TestSastScans.test_cwe798_scan) ... ok
test_cwe89_scan (repelsec.unit_tests.TestSastScans.test_cwe89_scan) ... ok

----------------------------------------------------------------------
Ran 27 tests in 0.003s

OK
```

7.5 Appendix 5: Performance Testing

Performance test 1:

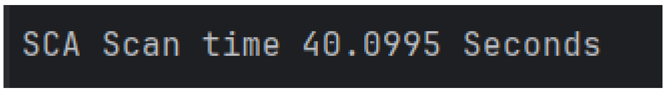


```
SAST Scan time 0.0302 Seconds
```

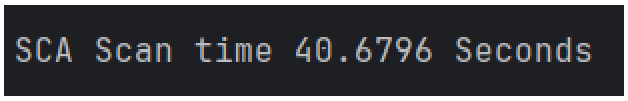


```
SAST Scan time 0.00339866 Seconds
```

Performance test 2:

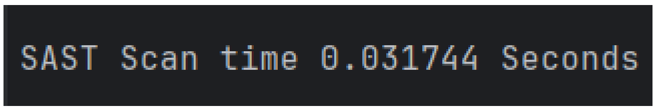


```
SCA Scan time 40.0995 Seconds
```

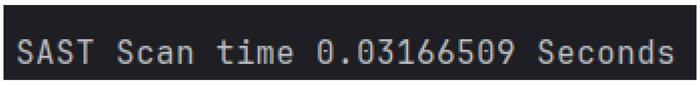


```
SCA Scan time 40.6796 Seconds
```

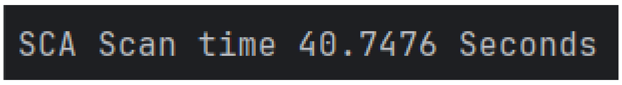
Performance test 3:



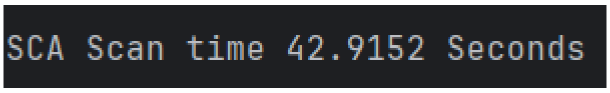
```
SAST Scan time 0.031744 Seconds
```



```
SAST Scan time 0.03166509 Seconds
```

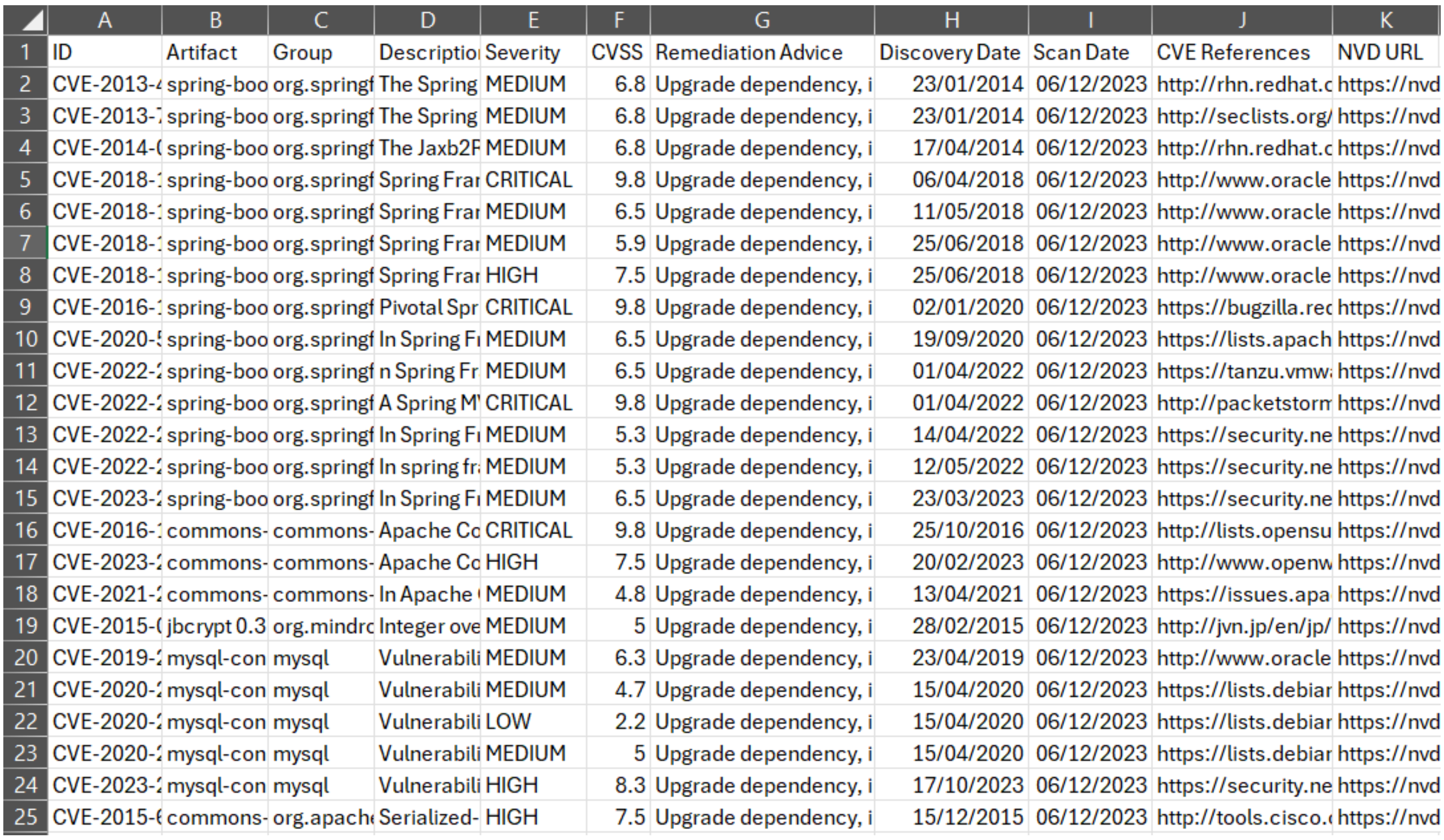


```
SCA Scan time 40.7476 Seconds
```



```
SCA Scan time 42.9152 Seconds
```

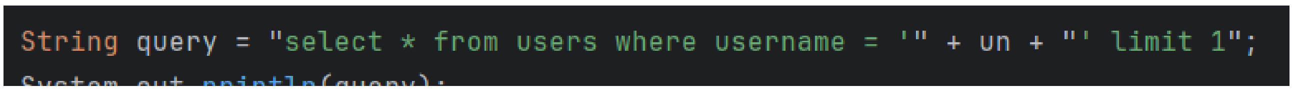
7.6 Appendix 6: CSV Output



	A	B	C	D	E	F	G	H	I	J	K
1	ID	Artifact	Group	Descriptio	Severity	CVSS	Remediation Advice	Discovery Date	Scan Date	CVE References	NVD URL
2	CVE-2013-4	spring-boo	org.springf	The Spring	MEDIUM	6.8	Upgrade dependency, i	23/01/2014	06/12/2023	http://rhn.redhat.c	https://nvd
3	CVE-2013-7	spring-boo	org.springf	The Spring	MEDIUM	6.8	Upgrade dependency, i	23/01/2014	06/12/2023	http://seclists.org/	https://nvd
4	CVE-2014-0	spring-boo	org.springf	The Jaxb2F	MEDIUM	6.8	Upgrade dependency, i	17/04/2014	06/12/2023	http://rhn.redhat.c	https://nvd
5	CVE-2018-1	spring-boo	org.springf	Spring Frar	CRITICAL	9.8	Upgrade dependency, i	06/04/2018	06/12/2023	http://www.oracle	https://nvd
6	CVE-2018-1	spring-boo	org.springf	Spring Frar	MEDIUM	6.5	Upgrade dependency, i	11/05/2018	06/12/2023	http://www.oracle	https://nvd
7	CVE-2018-1	spring-boo	org.springf	Spring Frar	MEDIUM	5.9	Upgrade dependency, i	25/06/2018	06/12/2023	http://www.oracle	https://nvd
8	CVE-2018-1	spring-boo	org.springf	Spring Frar	HIGH	7.5	Upgrade dependency, i	25/06/2018	06/12/2023	http://www.oracle	https://nvd
9	CVE-2016-1	spring-boo	org.springf	Pivotal Spr	CRITICAL	9.8	Upgrade dependency, i	02/01/2020	06/12/2023	https://bugzilla.rec	https://nvd
10	CVE-2020-5	spring-boo	org.springf	In Spring Fı	MEDIUM	6.5	Upgrade dependency, i	19/09/2020	06/12/2023	https://lists.apach	https://nvd
11	CVE-2022-2	spring-boo	org.springf	n Spring Fr	MEDIUM	6.5	Upgrade dependency, i	01/04/2022	06/12/2023	https://tanzu.vmw	https://nvd
12	CVE-2022-2	spring-boo	org.springf	A Spring M'	CRITICAL	9.8	Upgrade dependency, i	01/04/2022	06/12/2023	http://packetstorm	https://nvd
13	CVE-2022-2	spring-boo	org.springf	In Spring Fı	MEDIUM	5.3	Upgrade dependency, i	14/04/2022	06/12/2023	https://security.ne	https://nvd
14	CVE-2022-2	spring-boo	org.springf	In spring fr	MEDIUM	5.3	Upgrade dependency, i	12/05/2022	06/12/2023	https://security.ne	https://nvd
15	CVE-2023-2	spring-boo	org.springf	In Spring Fı	MEDIUM	6.5	Upgrade dependency, i	23/03/2023	06/12/2023	https://security.ne	https://nvd
16	CVE-2016-1	commons-	commons-	Apache Co	CRITICAL	9.8	Upgrade dependency, i	25/10/2016	06/12/2023	http://lists.opensu	https://nvd
17	CVE-2023-2	commons-	commons-	Apache Co	HIGH	7.5	Upgrade dependency, i	20/02/2023	06/12/2023	http://www.openw	https://nvd
18	CVE-2021-2	commons-	commons-	In Apache (	MEDIUM	4.8	Upgrade dependency, i	13/04/2021	06/12/2023	https://issues.apa	https://nvd
19	CVE-2015-(	jbcrypt 0.3	org.mindrc	Integer ove	MEDIUM	5	Upgrade dependency, i	28/02/2015	06/12/2023	http://jvn.jp/en/jp/	https://nvd
20	CVE-2019-2	mysql-con	mysql	Vulnerabili	MEDIUM	6.3	Upgrade dependency, i	23/04/2019	06/12/2023	http://www.oracle	https://nvd
21	CVE-2020-2	mysql-con	mysql	Vulnerabili	MEDIUM	4.7	Upgrade dependency, i	15/04/2020	06/12/2023	https://lists.debiar	https://nvd
22	CVE-2020-2	mysql-con	mysql	Vulnerabili	LOW	2.2	Upgrade dependency, i	15/04/2020	06/12/2023	https://lists.debiar	https://nvd
23	CVE-2020-2	mysql-con	mysql	Vulnerabili	MEDIUM	5	Upgrade dependency, i	15/04/2020	06/12/2023	https://lists.debiar	https://nvd
24	CVE-2023-2	mysql-con	mysql	Vulnerabili	HIGH	8.3	Upgrade dependency, i	17/10/2023	06/12/2023	https://security.ne	https://nvd
25	CVE-2015-6	commons-	org.apach	Serialized-	HIGH	7.5	Upgrade dependency, i	15/12/2015	06/12/2023	http://tools.cisco.	https://nvd

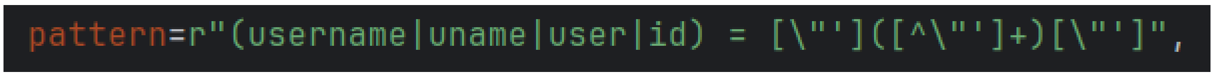
7.7 Appendix 7: Vulnado False Positive

False positive detected for CWE-798: Use of Hard-coded Credentials due to the pattern **username = ‘’**



```
String query = "select * from users where username = '" + un + "' limit 1";
System.out.println(query);
```

The regex was modified so that the term would not match within a string such as an SQL query.



```
pattern=r"(username|uname|user|id) = [\"'](['^\"']+)[\"']",
```