

Scanning open source software for zero-day vulnerabilities

Submitted in partial fulfillment of the requirements
of the degree of

BACHELOR OF ENGINEERING
In
COMPUTER ENGINEERING

By

Group No: 20

Furtado Jaden Ignatius Martin

Karna Shashank Sarsij

Panjwani Devika Anil

Raheja Hardik Deepak

Guide:

DR. TASNEEM MIRZA

(Associate Professor, Department of Computer Engineering, TSEC)



Computer Engineering Department
Thadomal Shahani Engineering College
University of Mumbai
2022-2023

Scanning open source software for zero-day vulnerabilities

Submitted in partial fulfillment of the requirements
of the degree of

BACHELOR OF ENGINEERING
In
COMPUTER ENGINEERING

By

Group No: 20

1902041 Furtado Jaden Ignatius Martin

1902070 Karna Shashank Sarsij

1902117 Panjwani Devika Anil

1902132 Raheja Hardik Deepak

Guide:

DR. TASNEEM MIRZA

(Associate Professor, Department of Computer Engineering, TSEC)



Computer Engineering Department
Thadomal Shahani Engineering College
University of Mumbai
2022-2023

CERTIFICATE

This is to certify that the project entitled “**Scanning open source software for zero day vulnerabilities**” is a bonafide work of

1902041 Furtado Jaden Ignatius Martin

1902070 Karna Shashank Sarsij

1902117 Panjwani Devika Anil

1902132 Raheja Hardik Deepak

Submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of “**BACHELOR OF ENGINEERING**” in “**COMPUTER ENGINEERING**”.

Dr. Tasneem Mirza
Guide

Dr. Tanuja Sarode
Head of Department

Dr. G. T. Thampi
Principal

Project Report Approval for B.E

Project report entitled ***Scanning open source software for zero day vulnerabilities*** by

1902041 Furtado Jaden Ignatius Martin

1902070 Karna Shashank Sarsij

1902117 Panjwani Devika Anil

1902132 Raheja Hardik Deepak

is approved for the degree of “**BACHELOR OF ENGINEERING**” in
“**COMPUTER ENGINEERING**”.

Examiners

1. _____

2. _____

Date:

Place:

Declaration

We declare that this written submission represents my ideas in my own words and where others 'ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

1) _____
Furtado Jaden Ignatius Martin- 1902041

2) _____
Karna Shashank Sarsij - 1902070

3) _____
Panjwani Devika Anil - 1902117

4) _____
Raheja Hardik Deepak - 1902132

Date:

Abstract

The use of open-source software has increased rapidly in recent years, and it has become a crucial part of many critical systems and applications. However, open-source software is not immune to vulnerabilities, including zero-day vulnerabilities. Zero-day vulnerabilities are a type of software vulnerability that are unknown to the software developer or vendor and therefore lack a patch or fix. The exploitation of zero-day vulnerabilities can lead to serious consequences, such as data breaches, system compromises, and financial losses. Therefore, it is essential to develop effective methodologies for scanning open-source software for zero-day vulnerabilities.

This project aims to develop a methodology for scanning software for vulnerabilities. Vulnerabilities are software weaknesses that can be exploited by attackers to gain unauthorized access to systems or data, causing significant damage. The proposed methodology involves a combination of automated and manual techniques for identifying and analyzing potential vulnerabilities. It includes identifying potential attack vectors, analyzing code patterns and dependencies, and performing manual code reviews to identify potential weaknesses. The project will also evaluate the effectiveness of the developed methodology by scanning a range of open-source software and identifying any zero-day vulnerabilities and other vulnerabilities.

The successful completion of this project will contribute to the improvement of the security of open-source software. The developed methodology will help identify and address potential zero-day vulnerabilities before they can be exploited. The project's outcomes will be essential for enhancing the security of software systems by identifying and addressing potential vulnerabilities before they can be exploited and it also can be used to enhance the security of critical systems and applications that rely on open-source software. Overall, this project is crucial for ensuring the security and integrity of software systems in an increasingly digital world.

In this project we were able to scan over 3000 open source repositories available on GitHub as well as over 1000 apps. We were able to identify over 12,000 vulnerabilities and were able to find critical flaws exposing millions of users worldwide.

Table of Content

Chapter 1	Introduction	1
	1.1 Introduction	1
	1.2 Problem Statement and Objectives	2
	1.3 Scope	3
Chapter 2	Review of Literature	4
	2.1 Domain Explanation	4
	2.2 Review of Existing System	5
	2.3 Limitations of Existing System/Research Gaps	6
Chapter 3	Proposed System	8
	3.1 Design Details	8
	3.2 Methodology	14
Chapter 4	Implementation Details	21
	4.1 Experimental Setup	21
	4.2 Software and Hardware Setup	24
Chapter 5	Results and Discussion	28
	5.1 Performance Evaluation Parameters	28
	5.2 Implementation Results	29
	5.3 Results Discussion	32
Chapter 6	Conclusion and Future Work	33
References		34
Acknowledgement		36

List of Figures

Figure No.	Description	Page No.
Figure 3.1	Stages of a compiler	8
Figure 3.2	Working of Static Code Analysis	10
Figure 3.3	High level Architecture diagram	10
Figure 3.4	Example of GitHub action workflow	12
Figure 3.5	Working of Celery	16
Figure 3.6	Architecture of Celery	16
Figure 3.7	File Manager Interface	17
Figure 3.8	Options available to add files via upload, link, Git	18
Figure 3.9	User Permission control via File Manager for Access Control	18
Figure 3.10	Actions available with File Manager	19
Figure 3.11	Text Editor Present on File Manager	19
Figure 4.1	CLI tool for automation	22
Figure 4.2	Production of results as a JSON File	22
Figure 4.3	List of vulnerabilities detected via Scan	23

Figure No.	Description	Page No.
Figure 4.4	Detailed expansion of a particular vulnerability showing code snippet and type of vulnerability (XSS)	23
Figure 4.5	Common Weakness Enumeration (CWEs) distribution found in the given scan	23
Figure 4.6	List of open-source repositories scanned	25
Figure 4.7	List of reports being produced while scanning	26
Figure 4.8	Results of scans	26
Figure 4.9	List of vulnerabilities	27
Figure 5.1	Distribution of vulnerabilities represented on a donut graph in the dashboard	29
Figure 5.2	Code snippet along with location of vulnerability on the dashboard	29
Figure 5.3	Buffer overflow being detected by the scan on the dashboard along with code snippet	30
Figure 5.4	A snippet of vulnerable code showing Hardcoded credentials	30
Figure 5.5	Benchmarks of Semgrep run over several repositories over years	31
Figure 5.6	Semgrep scan time as compared to other Python analysis tools on Django repository	31

Chapter 1

Introduction

1.1 Introduction

As software continues to play an increasingly critical role in modern society, the security of software systems becomes a top priority. Vulnerabilities in software can be exploited by attackers to gain unauthorized access to systems or data, causing significant damage. Zero-day vulnerabilities are a type of software vulnerability that are unknown to the software developer or vendor and therefore lack a patch or fix. The exploitation of zero-day vulnerabilities can lead to serious consequences, such as data breaches, system compromises, and financial losses. Therefore, it is essential to develop effective methodologies for scanning open-source software for zero-day vulnerability and other vulnerabilities.

This project aims to develop a methodology for scanning software for vulnerabilities. This project aims to develop such a methodology using a combination of automated and manual techniques. The methodology will involve identifying potential attack vectors, analyzing code patterns and dependencies, and performing manual code reviews to identify potential weaknesses [1]. The project will require a deep understanding of software vulnerabilities, software security, and software development. The project team will need to have expertise in programming languages, software analysis tools, and software security testing. The successful

completion of this project will contribute to the improvement of the security of software systems in an increasingly digital world.

The increasing reliance on software in critical systems and applications highlights the importance of effective vulnerability scanning techniques. The proposed methodology can aid in improving the security of software products by identifying and addressing potential vulnerabilities before they can be exploited. The outcomes of this project can be used by software developers, vendors, and security researchers to enhance the security and integrity of software systems in an increasingly digital world.

1.2 Problem Statement & Objectives

The aim of this project is to develop a methodology for scanning software for vulnerabilities that combines automated and manual techniques to enhance the security of software systems. The proposed methodology will involve a comprehensive analysis of software code to identify potential vulnerabilities that can be exploited by attackers.

Identify potential attack vectors and common vulnerability patterns: The first objective of this project is to identify potential attack vectors and common vulnerability patterns that can be exploited by attackers. This will involve analyzing past attacks and vulnerability reports to determine the most common types of software vulnerabilities.

Analyze code patterns and dependencies: The second objective is to analyze code patterns and dependencies to identify potential vulnerabilities that may not be apparent through automated techniques. This will involve examining the software code to identify common coding errors and weaknesses that can be exploited by attackers.

Use automated tools and scripts: The third objective is to use automated tools and scripts to scan software for common vulnerabilities. This will involve making use of tools that can detect common vulnerabilities such as buffer overflows, injection attacks, and authentication bypasses.

Test the effectiveness of the developed methodology: The fourth objective is to test the effectiveness of the developed methodology by scanning a range of software products and identifying any vulnerabilities. This will involve scanning software from different domains and categories to determine the efficacy of the developed methodology.

Contribute to the development of industry-standard vulnerability scanning techniques: The fifth objective is to contribute to the development of industry-standard vulnerability scanning

techniques by sharing the outcomes of the project with the wider security community. This will involve publishing the results of the project in academic journals and presenting them at relevant conferences.

Provide recommendations for enhancing the security of software systems: The sixth objective is to provide recommendations for enhancing the security of software systems based on the vulnerabilities identified during the project. This will involve synthesizing the results of the project and developing practical recommendations for software developers, vendors, and end-users to enhance the security of their software systems.

1.3 Scope

The project will involve the development of automated tools and scripts for vulnerability scanning, as well as the implementation of manual code review techniques to identify potential weaknesses in software code. The report shall include the type of vulnerability, its severity and possibly a resolution. The tool is strong enough to find zero day vulnerabilities too. Some of the common vulnerabilities it can easily find in the application are: SQL injection, Weak passwords, Missing data encryption, Malware, Virus, Ransomware and many more.

The methodology for scanning open source software for zero-day vulnerabilities could involve using a combination of automated tools and manual analysis to identify any potential vulnerabilities. Automated tools can scan the source code of the software to identify potential vulnerabilities, while manual analysis can be used to validate the results and identify any additional vulnerabilities that may have been missed by the automated tools.

Once the scan is complete, a detailed report should be generated. The report should include the vulnerabilities identified, their severity, and any recommendations for remediation. The report should also include a risk assessment, highlighting the potential impact of the vulnerabilities if they were to be exploited by attackers. The report should be shared with the relevant stakeholders, including developers and security teams, so that they can take the necessary steps to address the vulnerabilities and reduce the risk of a potential attack.

The project will require a deep understanding of software vulnerabilities, software security, and software development. The project team will need to have expertise in programming

languages, software analysis tools, and software security testing. The project will also require access to a range of software products for testing the developed methodology.

The project outcomes will be relevant to software developers, vendors, and end-users who rely on software systems for various purposes. The recommendations provided by the project will enable these stakeholders to enhance the security of their software systems and reduce the risk of potential security breaches. The project outcomes will also contribute to the development of industry-standard vulnerability scanning techniques and promote best practices in software security.

Chapter 2

Review of Literature

2.1 Domain Explanation

The domain of this project is cybersecurity, identifying and addressing potential vulnerabilities in software systems with a specific focus on identifying and addressing zero-day vulnerabilities in open source software. Zero-day vulnerabilities are security weaknesses in software systems that are unknown to the software developer or vendor and, therefore, have no patch or fix available. These vulnerabilities are highly sought after by attackers because they provide a unique opportunity to exploit software systems without detection.

Open source software, which is software that is freely available for use and modification by anyone, is widely used in various domains, including finance, healthcare, e-commerce, and government. Due to the open and collaborative nature of open source software development, these systems are at higher risk of vulnerabilities that can go undetected for long periods.

Cybersecurity is critical in protecting against potential attacks and maintaining the confidentiality, integrity, and availability of sensitive information. This project aims to contribute to the cybersecurity domain by developing a methodology for identifying zero-day vulnerabilities in open source software. The project outcomes will be relevant to software

developers, vendors, and end-users who rely on open source software systems and need to ensure their security against potential attacks.

The domain of cybersecurity is constantly evolving, and attackers are continually developing new techniques and methods to exploit vulnerabilities in software systems. Therefore, it is essential to develop new and improved techniques for identifying and addressing zero-day vulnerabilities in open source software. This project aims to contribute to the ongoing efforts to enhance cybersecurity and promote best practices in identifying and addressing zero-day vulnerabilities.

2.2 Review of Existing Systems

Following are a few of the existing tools used for finding vulnerabilities in codes.

Enlightn [5] uses a variety of techniques to analyze and optimize web applications for security, performance, and cost. Some of the key techniques used by the tool include:

Automated scanning: Enlightn uses automated scanning to identify potential security vulnerabilities and performance bottlenecks in web applications. The tool scans for common issues such as OWASP top 10 risks, slow database queries, and inefficient code.

Intelligent recommendations: Based on the results of the scan, Enlightn provides intelligent recommendations for fixing potential issues. These recommendations may include code snippets, configuration changes, or other optimizations to improve security and performance.

Integration with DevOps tools: Enlightn integrates with popular DevOps tools such as Jenkins, GitLab, and GitHub to provide a seamless workflow for developers and DevOps teams. Overall, Enlightn uses a combination of automated scanning, intelligent recommendations, integration with DevOps tools, and detailed reporting to optimize web applications for security, performance, and cost.

SpotBugs [6] is a static code analysis tool that identifies potential software bugs in Java programs. It is a successor to the well-known FindBugs tool and uses similar techniques to identify issues such as null pointer dereferences, infinite loops, and other potential security vulnerabilities.

SpotBugs performs its analysis by inspecting the compiled bytecode of Java programs. It can identify bugs and potential issues that may not be apparent during code review or testing. The tool provides detailed reports that highlight potential issues and recommended fixes. These reports can be used by developers to improve the quality and security of their code.

SpotBugs can be integrated with popular development environments such as Eclipse and IntelliJ IDEA, as well as with build tools such as Apache Maven and Gradle. This allows developers to seamlessly integrate static code analysis into their development workflow.

Overall, SpotBugs is a powerful tool for identifying potential bugs and security vulnerabilities in Java programs. Its integration with popular development environments and build tools make it a convenient choice for developers looking to improve the quality and security of their code.

Flawfinder [7] is an open-source tool used for identifying potential security vulnerabilities in software source code. It is designed to be used by software developers and security analysts to identify common coding mistakes and security issues that can lead to exploits, hacking, and other security threats.

Flawfinder analyzes source code in various programming languages, including C, C++, Java, and Python, to identify potential security issues. It searches for keywords, functions, and patterns that are known to be associated with security vulnerabilities, such as buffer overflows, format string vulnerabilities, and SQL injection attacks.

One of the key benefits of Flawfinder is that it is simple to use and can quickly scan large codebases for potential security vulnerabilities. It produces an easy-to-read report that identifies each vulnerability and provides recommendations for how to fix it.

However, Flawfinder does have some limitations. It relies on known patterns and keywords to identify security vulnerabilities, which means that it may miss some newer or more obscure vulnerabilities. Additionally, it is not a replacement for a comprehensive security audit, and it cannot identify all possible security issues in a codebase.

Despite these limitations, Flawfinder is a valuable tool for developers and security analysts looking to quickly and easily identify potential security vulnerabilities in their code. Its ease of use and ability to quickly identify common security issues make it a valuable addition to any software development or security toolkit.

2.3 Limitations of Existing System/ Research Gaps

Vulnerability scanning tools are an essential component in detecting and preventing security breaches in software systems. However, these tools have several limitations that must be considered to ensure effective vulnerability management.

One of the main limitations of vulnerability scanning tools is the high rate of false positives. False positives occur when the tool incorrectly identifies code patterns as vulnerabilities when they are actually safe coding practices. This can be time-consuming for developers who have to sift through a large number of false positives to identify true vulnerabilities. To mitigate this, it is important to use vulnerability scanning tools in conjunction with manual code review and other security measures [8].

Another limitation of vulnerability scanning tools is their limited coverage. These tools can only detect vulnerabilities that are known to the tool, and may not be able to identify new and novel vulnerabilities. This can leave systems at risk, particularly if attackers are using previously unknown vulnerabilities to exploit systems. To address this limitation, it is important to stay up-to-date on the latest security threats and vulnerabilities, and to use multiple vulnerability scanning tools to increase the chances of detection.

The accuracy of vulnerability scanning tools is another limitation to consider. These tools may not always accurately identify vulnerabilities, particularly for complex systems or for vulnerabilities that are difficult to detect. To address this, it is important to use multiple vulnerability scanning tools and to verify identified vulnerabilities through manual code review and penetration testing.

In addition, vulnerability scanning tools may have limited scope and may only focus on specific types of vulnerabilities or programming languages. This can leave other vulnerabilities undetected and can give a false sense of security. It is important to use multiple vulnerability scanning tools and to ensure that they cover a wide range of vulnerabilities and programming languages [2] [3].

Finally, vulnerability scanning tools may not be able to detect zero-day vulnerabilities, which are vulnerabilities that have not yet been discovered or publicly disclosed. This is a significant limitation as attackers often use zero-day vulnerabilities to exploit systems. To address this limitation, it is important to stay up-to-date on the latest security threats and to use multiple vulnerability scanning tools that can detect both known and unknown vulnerabilities

Chapter 3

Proposed System

3.1 Design Details

The premise of our proposed system is on conducting the static code analysis of code so as to look for and find vulnerabilities in code. At a deeper level, it can be explained as by Figure 3.1

Following are the steps involved in a compiler:

- **Lexical Analysis**

The first phase of the scanner works as a text scanner. This phase scans the source code as a stream of characters and converts it into meaningful lexemes. Lexical analyzer represents these lexemes in the form of tokens as:

<token-name, attribute-value>

- **Syntax Analysis**

The next phase is called the syntax analysis or parsing. It takes the token produced by lexical analysis as input and generates a parse tree (or syntax tree). In this phase, token arrangements are checked against the source code grammar, i.e. the parser checks if the expression made by the tokens is syntactically correct.

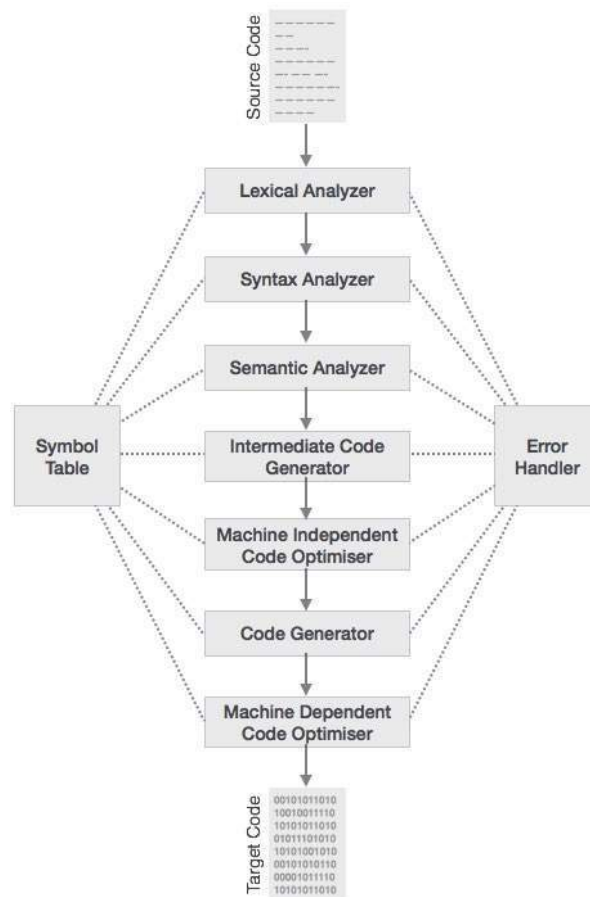


Figure 3.1 Stages of a compiler

- Semantic Analysis

Semantic analysis checks whether the parse tree constructed follows the rules of language. For example, assignment of values is between compatible data types, and adding string to an integer. Also, the semantic analyzer keeps track of identifiers, their types and expressions; whether identifiers are declared before use or not etc. The semantic analyzer produces an annotated syntax tree as an output.

- Intermediate Code Generation

After semantic analysis the compiler generates an intermediate code of the source code for the target machine. It represents a program for some abstract machine. It is in between the high-level language and the machine language. This intermediate code should be generated in such a way that it makes it easier to be translated into the target machine code.

- Code Optimization

The next phase does code optimization of the intermediate code. Optimization can be assumed as something that removes unnecessary code lines, and arranges the sequence of statements in order to speed up the program execution without wasting resources such as

CPU, memory, etc.

- Code Generation

In this phase, the code generator takes the optimized representation of the intermediate code and maps it to the target machine language. The code generator translates the intermediate code into a sequence of (generally) relocatable machine code. Sequence of instructions of machine code performs the task as the intermediate code would do.

- Symbol Table

It is a data-structure maintained throughout all the phases of a compiler. All the identifier's names along with their types are stored here. The symbol table makes it easier for the compiler to quickly search the identifier record and retrieve it. The symbol table is also used for scope management.

While it is certainly possible to find vulnerabilities in the code at each of these individual phases of the compiler, the phase we decided to focus on is the semantic analysis phase as we can use regular expressions to analyze the control flow graphs for vulnerabilities of known vulnerability patterns.

A high level flow graph of how Static code analysis works is shown below:

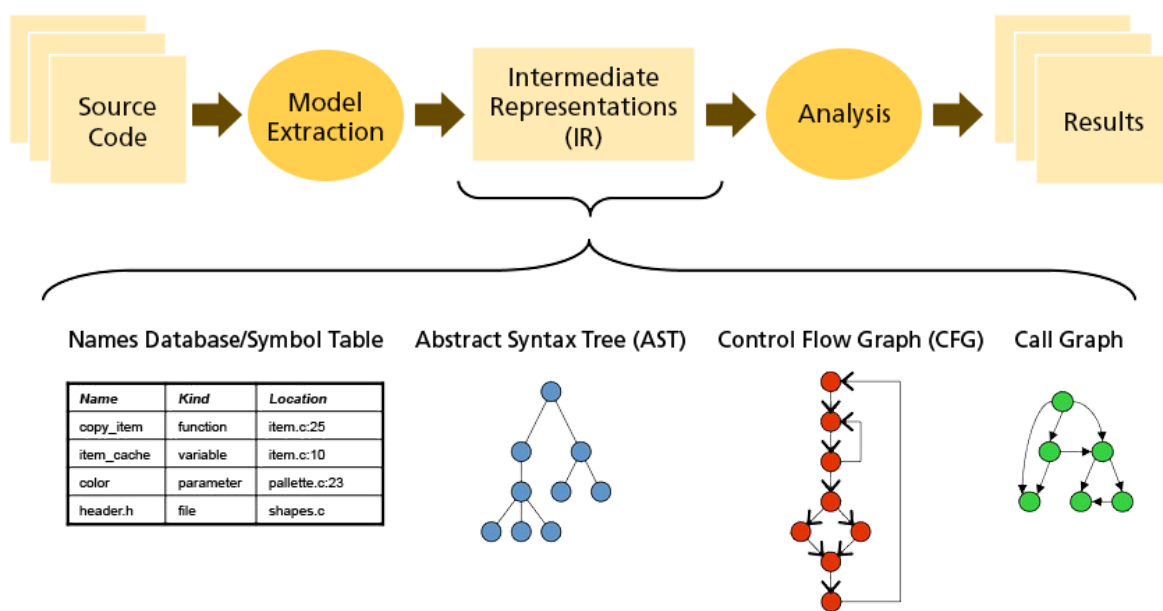


Figure 3.2 Working of static code analysis

Our High level architecture diagram is displayed in Figure 3.3

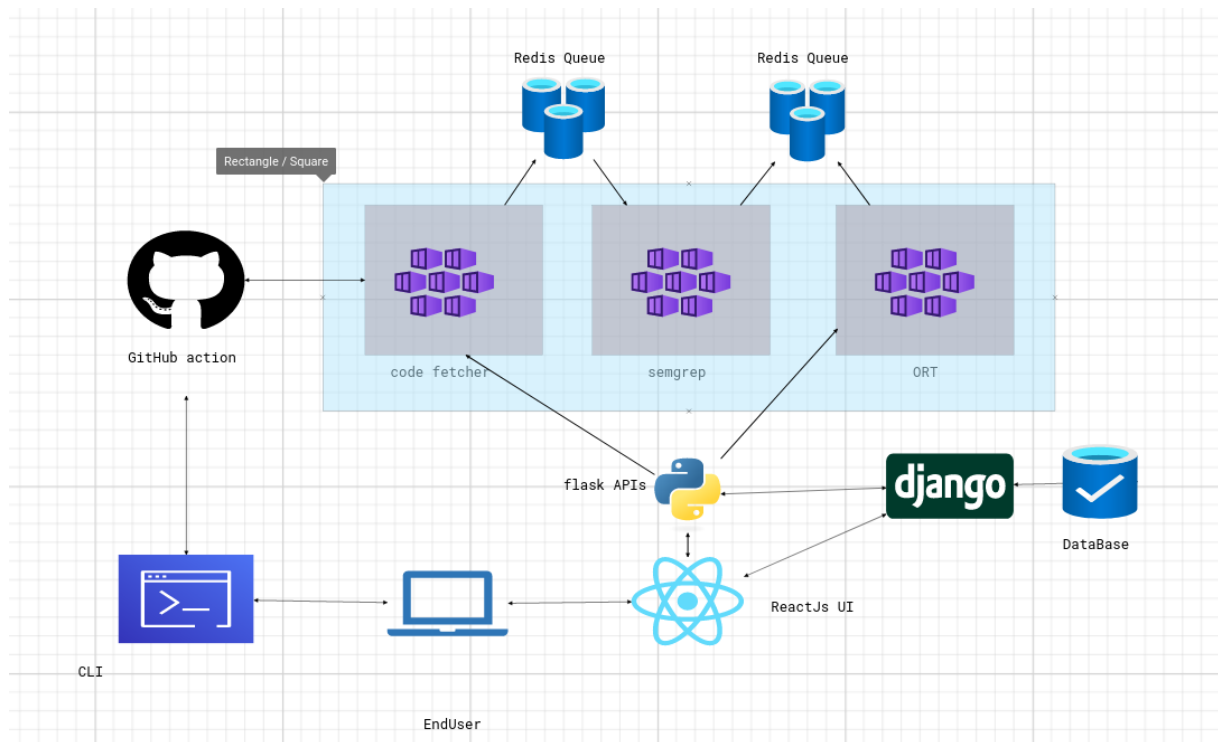


Figure 3.3 High level architecture diagram

We can divide the project into the following modules based on the functionality:

Code Scanner module:

The code scanner module is the blue coloured block shown in the above architecture diagram. It integrates two or more submodules in it, namely the code fetcher, Semgrep, ORT and other scanners which we may want to use. The purpose of this module is to put all the code scanners that a given user may want to use for their use case under one roof for seamless integration.

- Code fetcher: Fetch the code from the remote location such as GitHub, GitLab, GitPod, etc
- Semgrep [4]: Semgrep or semantic grepping is an open source static code analysis tool
- ORT: Open source review toolkit, or ORT is an open source framework for determining the compliance of open source repositories and packages along with their dependencies.

Flask API wrappers [17]:

These are a bunch of APIs written in flask that act as a wrapper around the Code Scanner module and allow us to access the functionality using REST API calls. The advantage of REST API means that we can implement any form of authentication for access control at this

layer as well as call the code scanner module using a GUI or Graphical User Interface remotely

Frontend:

The frontend written in PHP, HTML, CSS and JS is used to interact with the Flask APIs and the Django Backend to fetch relevant data from our current scans as well as our past scans

Django Backend:

The Django Backend is used for any form of processor/computationally intensive work that we may need to do, such as data analysis on our results, fetching a history of scans, fetching vulnerabilities that we have found in bulk, etc. Django is a free and open-source, Python-based web framework that follows the model–template–views architectural pattern. It is maintained by the Django Software Foundation, an independent organization established in the US as a 501 non-profit.

GitHub Actions [16]:

GitHub actions allow our solution to be integrated in CI/CD pipelines (Continuous Integration, Continuous Deployment), thus allowing a greater deal of automation to the code scanner.

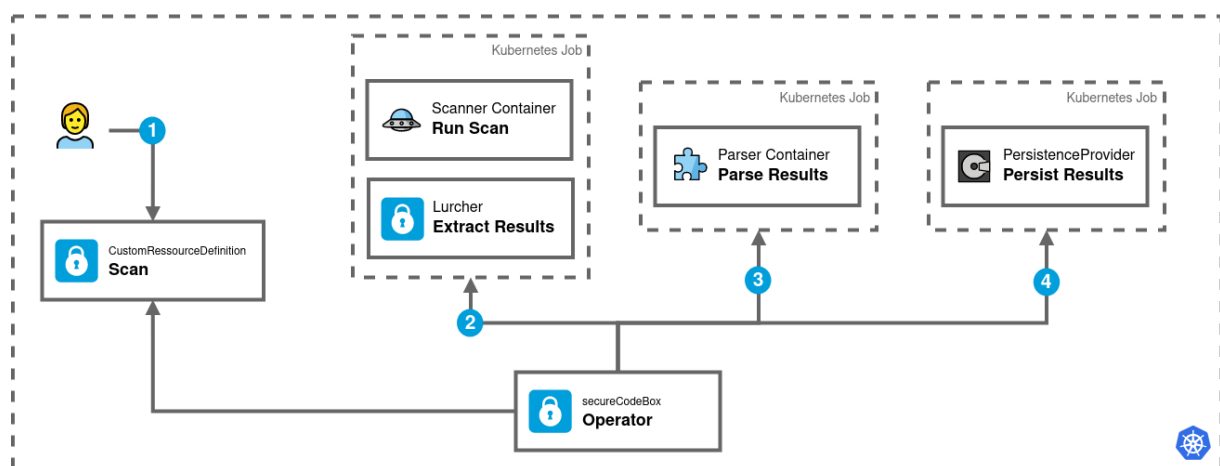


Figure 3.4 Example of GitHub action workflow

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows

that build and test every pull request to your repository, or deploy merged pull requests to production.

Database:

The Database is used to store results from our scans. The main purpose being to be able to analyze the historic results of scans on a particular piece of code as well as to be able to prevent duplicated from existing

CLI:

The CLI or Command Line Interface is a standalone tool that we developed that would allow a user to access ScanRE via their Command Line.

FileManager [9]:

We have included a file manager written in PHP to show that our tool could be used to facilitate remote development. This shall ease the user to select a directory or a file for scanning. This makes the process of scanning available with GUI.

DefectDojo [13]:

DefectDojo is a vulnerability orchestration and management tool written in Django. We use it to demonstrate that our tool can be integrated with other existing tools in a company, thereby saving cost and minimizing wastage of time and effort. We also use it to visualize our findings and prove their validity.

Docker [18]:

Docker is a platform that allows developers to easily create, deploy, and run applications in containers. Containers are lightweight, standalone packages that contain all the necessary dependencies and configurations to run the application.

Docker Swarm is a clustering and orchestration tool for Docker containers that allows you to manage a group of Docker hosts as a single virtual system. It enables you to deploy and manage containers across multiple Docker hosts and provides features such as load balancing, service discovery, and automatic container placement.

Steps involved in implementing Docker Swarm-

1. Create a Docker Swarm cluster: The first step is to create a Docker Swarm cluster by initializing a Swarm on one of the nodes and then joining the other nodes to the Swarm.
2. Create a Docker image for the code scanner: Docker images for Semgrep and ORT scanner along with their dependencies are used. These images can be built using a Dockerfile that specifies the scanner and its dependencies.
3. Create a Docker service for the code scanner: Docker service that uses these images to run the code scanner is created. The service can be scaled up or down dynamically based on the workload.
4. Create a Docker service for the code fetcher: A Docker service for code fetcher which fetches the code from remote locations such as GitHub, Gitlab, etc.
5. Use asynchronous messaging to coordinate the code scanner and fetcher: A message queue is used to coordinate the code scanner and fetcher. The fetcher pushes the code to the message queue, and the scanner subscribes to the message queue to fetch the code and scan it. This approach enables asynchronous processing and can improve the overall performance.
6. Monitor and manage the Docker Swarm cluster: Finally, Docker Swarm's built-in monitoring and management features are used to monitor the cluster's performance and resource utilization.

3.2 Methodology

Code Scanner module:

We Started by working on the Code Scanner module. We decided that in order to ensure modularity and cohesion of our code as well as to ensure ease of future development, it would be easier to write this module as being a class in python.

The class has a constructor that accepts a repository link, repository name, scan result path and a clone path. We use this information at various points in our flow. Along with this, our scanner class has a number of other asynchronous methods, namely, getCode, scanCode, cleanUp, importScanData and scanFiles. asyncio is a library to write concurrent code using the async/await syntax.

asyncio is used as a foundation for multiple Python asynchronous frameworks that provide high-performance network and web-servers, database connection libraries, distributed task queues, etc. Asyncio is often a perfect fit for IO-bound and high-level structured network code.

Code Fetcher

Once our class is initialized, we then call the getCode method, which acts as our code fetcher module. We run a git clone command to clone a remote repository locally, thus fetching the code.

Code Scanner

The scanning of the code is handled by this module as shown here. For the purposes of demonstration, we have used Semgrep to do the static code analysis of our target repositories.

Semgrep or Semgrep CLI is a free open-source static code analysis tool developed by Return To Corporation and open-source contributors. It has stable support for Go, Java, JavaScript, JSON, Python, and Ruby. It has experimental support for eleven other languages, as well as a language agnostic mode.

Deep Scan:

The deepScan module takes in a repository or directory and scans all its dependencies as well, this is achieved with the help of ORT downloader which can download the dependencies across various tech stacks using a single command.

The OSS Review Toolkit (ORT) is a FOSS policy automation and orchestration toolkit which you can use to manage your (open source) software dependencies in a strategic, safe and efficient manner.

Utility methods

These were methods, that while not strictly required by our code, were required to ensure that we do not accumulate garbage and so used for cleanup after a scan

As shown, we delete the repository of code after we have completed the scan.

Finally, we would want to conduct a proper analysis of our scan results and so, we have written a method to send the results of our scan to our vulnerability management and security orchestration tool.

Flask APIs:

We then wrote a route in Flask, to accept a new request to scan a piece of code. It accepts both GET and POST requests. The output is a simple JSON result if all executes as planned. In order to ensure that we can execute code asynchronously, we have used celery.

We delay the execution of the rest of the code until the completion of our previous line. This however, is independent of the flask module. Celery [12] is an open source asynchronous task queue or job queue which is based on distributed message passing. While it supports scheduling, its focus is on operations in real time. We have used RedisDB as our Queue to ensure FIFO execution of jobs. The code fetcher submodule of our code can be thought of as the producer and the code scanner submodule can be thought of as being a client.

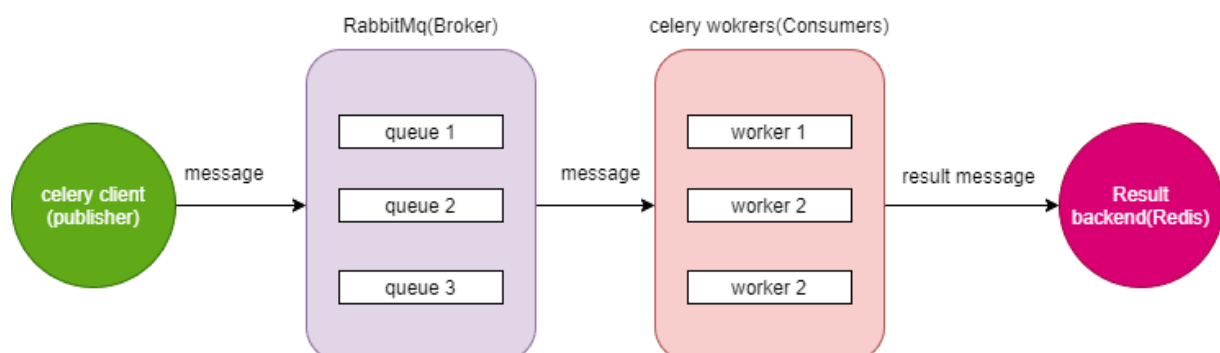


Figure 3.5 Working of Celery

Since the two are independent of each other in terms of functionality, it makes more sense to think of them as being a producer consumer problem.

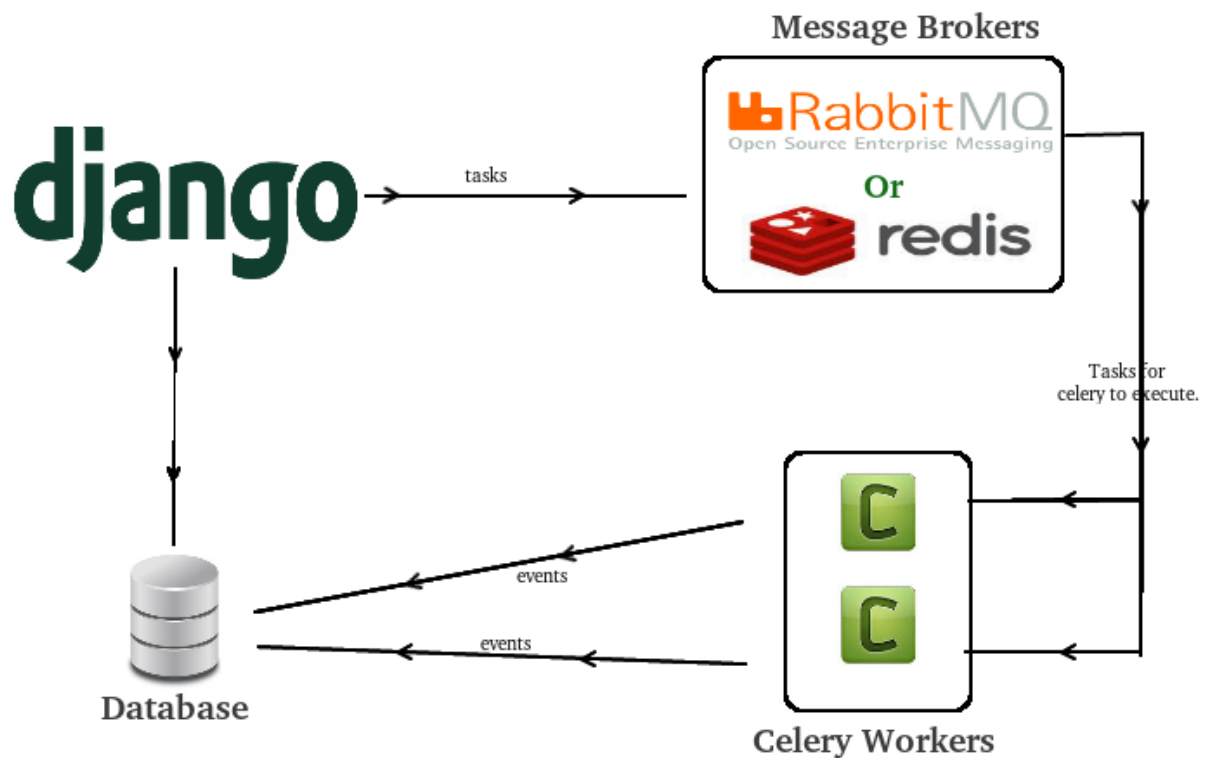


Figure 3.6 Architecture of Celery

Scaffold.sh and start.sh

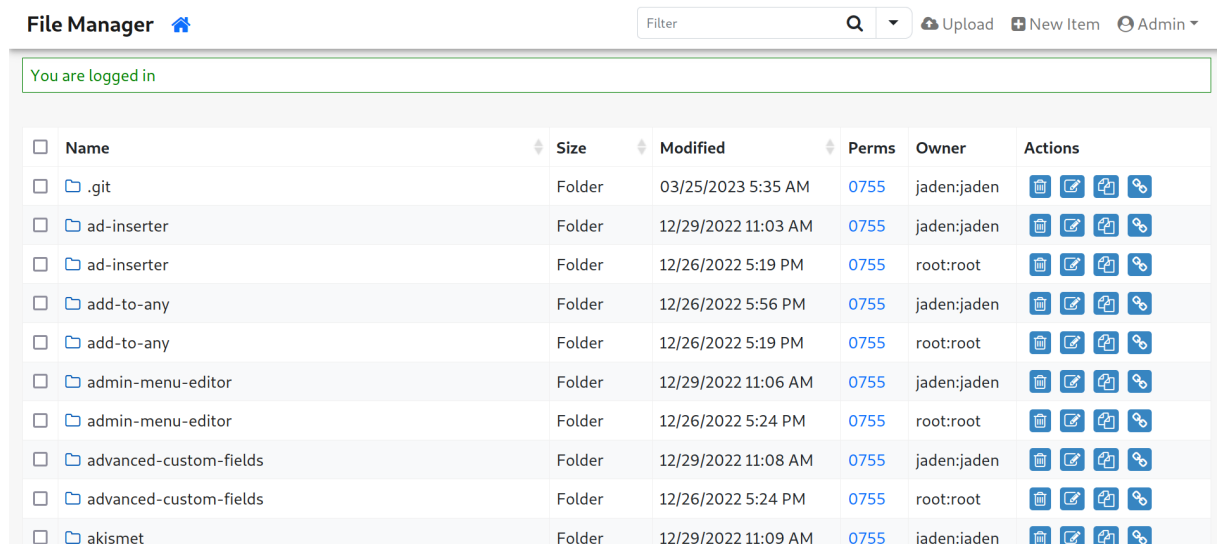
To better ensure that a new user does not have to suffer the overhead of having to setup our project from scratch, we created two bash scripts to automate the running of setup commands.

It pulls the semgrep image from the docker repository and installs all python dependencies using pip.

Start.sh is used to perform sanity checks such as seeing if a user has set a .env file containing the necessary environment variables for our project. If not, we throw an error. If we have all conditions met, we can start celery and our docker container.

Frontend:

We based our frontend on the tinyfile manager which is a simple file manager written in PHP. Open Source, light and extremely simple, It is a single executable file with no dependencies. It allows us to create, edit, copy, move, download your files easily, everywhere, every time and can be used as your personal cloud.

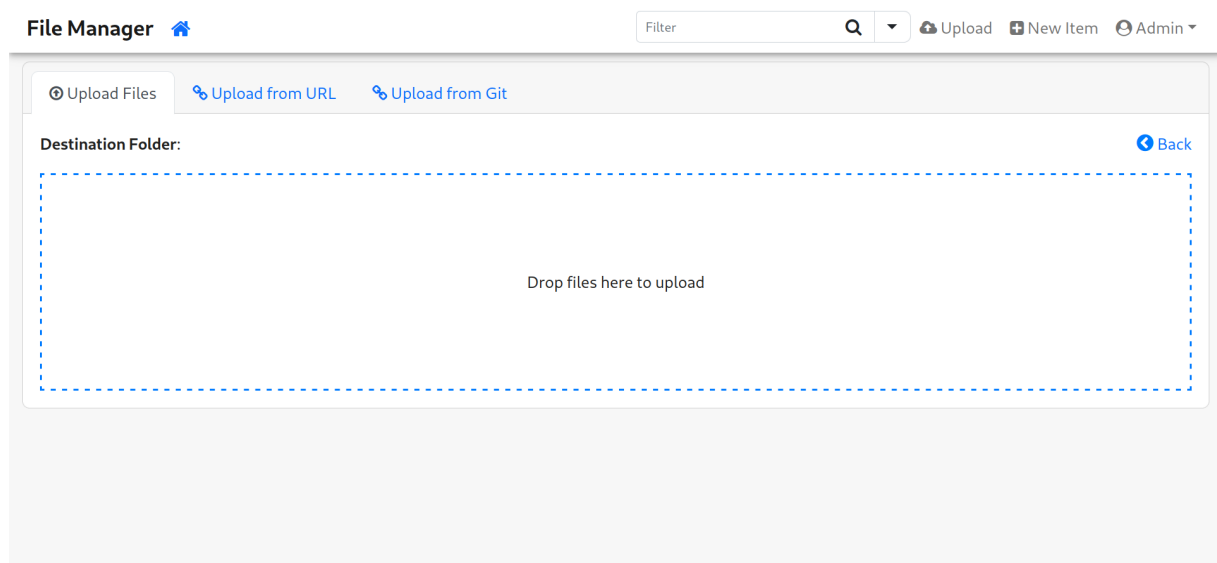


The screenshot shows the File Manager interface. At the top, there's a header with "File Manager" and a home icon. To the right is a search bar labeled "Filter" and buttons for "Upload", "New Item", and "Admin". Below the header, a green message says "You are logged in". The main area is a table with columns: Name, Size, Modified, Perms, Owner, and Actions. The table lists several folders, including .git, ad-inserter, add-to-any, admin-menu-editor, advanced-custom-fields, and akismet. Each row has a checkbox on the left and a set of action icons (delete, edit, copy, link) on the right.

<input type="checkbox"/>	Name	Size	Modified	Perms	Owner	Actions
<input type="checkbox"/>	.git	Folder	03/25/2023 5:35 AM	0755	jaden:jaden	
<input type="checkbox"/>	ad-inserter	Folder	12/29/2022 11:03 AM	0755	jaden:jaden	
<input type="checkbox"/>	ad-inserter	Folder	12/26/2022 5:19 PM	0755	root:root	
<input type="checkbox"/>	add-to-any	Folder	12/26/2022 5:56 PM	0755	jaden:jaden	
<input type="checkbox"/>	add-to-any	Folder	12/26/2022 5:19 PM	0755	root:root	
<input type="checkbox"/>	admin-menu-editor	Folder	12/29/2022 11:06 AM	0755	jaden:jaden	
<input type="checkbox"/>	admin-menu-editor	Folder	12/26/2022 5:24 PM	0755	root:root	
<input type="checkbox"/>	advanced-custom-fields	Folder	12/29/2022 11:08 AM	0755	jaden:jaden	
<input type="checkbox"/>	advanced-custom-fields	Folder	12/26/2022 5:24 PM	0755	root:root	
<input type="checkbox"/>	akismet	Folder	12/29/2022 11:09 AM	0755	jaden:jaden	

Figure 3.7 File Manager Interface

Ajax Upload, Ability to drag & drop, multiple files upload and upload using url with file extensions filter. We have added an additional functionality of uploading files from GitHub



The screenshot shows the File Manager interface with the upload options menu open. At the top, there's a header with "File Manager" and a home icon. To the right is a search bar labeled "Filter" and buttons for "Upload", "New Item", and "Admin". Below the header, a green message says "You are logged in". The main area is a form with three tabs: "Upload Files", "Upload from URL", and "Upload from Git". The "Upload Files" tab is selected. Below the tabs, there's a section labeled "Destination Folder:" with a "Back" button. A large dashed blue box is centered in the form, with the text "Drop files here to upload" inside it.

<input type="checkbox"/>	Name	Size	Modified	Perms	Owner	Actions
<input type="checkbox"/>	.git	Folder	03/25/2023 5:35 AM	0755	jaden:jaden	
<input type="checkbox"/>	ad-inserter	Folder	12/29/2022 11:03 AM	0755	jaden:jaden	
<input type="checkbox"/>	ad-inserter	Folder	12/26/2022 5:19 PM	0755	root:root	
<input type="checkbox"/>	add-to-any	Folder	12/26/2022 5:56 PM	0755	jaden:jaden	
<input type="checkbox"/>	add-to-any	Folder	12/26/2022 5:19 PM	0755	root:root	
<input type="checkbox"/>	admin-menu-editor	Folder	12/29/2022 11:06 AM	0755	jaden:jaden	
<input type="checkbox"/>	admin-menu-editor	Folder	12/26/2022 5:24 PM	0755	root:root	
<input type="checkbox"/>	advanced-custom-fields	Folder	12/29/2022 11:08 AM	0755	jaden:jaden	
<input type="checkbox"/>	advanced-custom-fields	Folder	12/26/2022 5:24 PM	0755	root:root	
<input type="checkbox"/>	akismet	Folder	12/29/2022 11:09 AM	0755	jaden:jaden	

Figure 3.8 Options available to add files via upload, links, Git

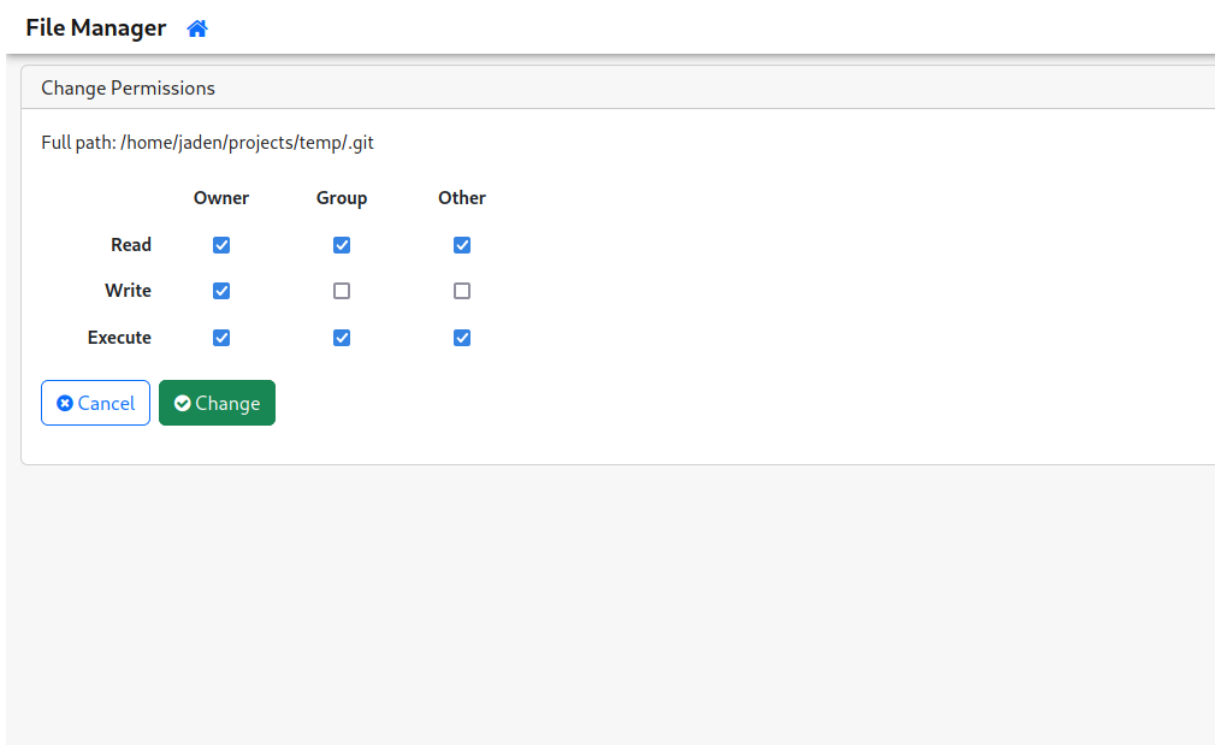


Figure 3.9 User Permission control via File Manager for Access Control

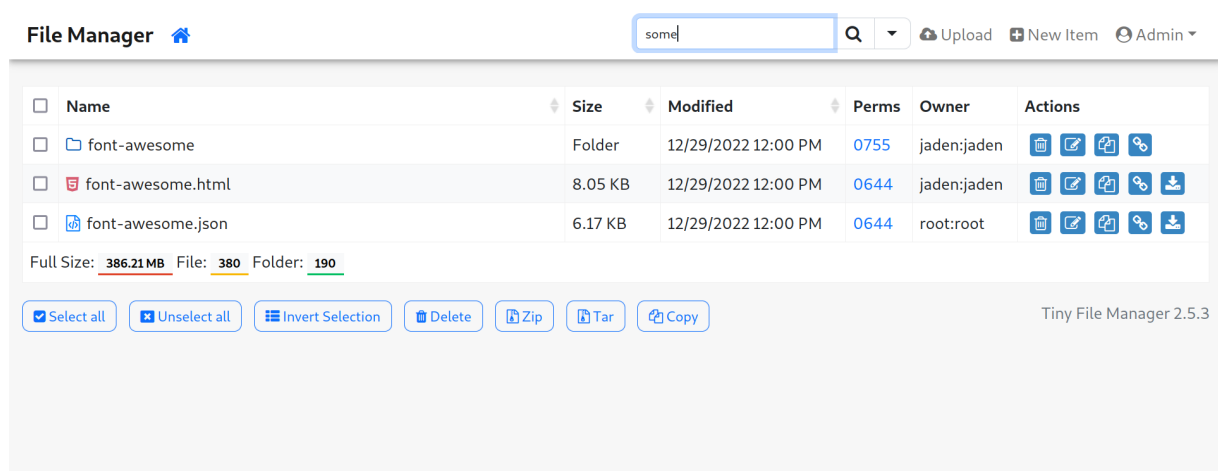


Figure 3.10 Actions available with File Manager such as search, filter, search by extension

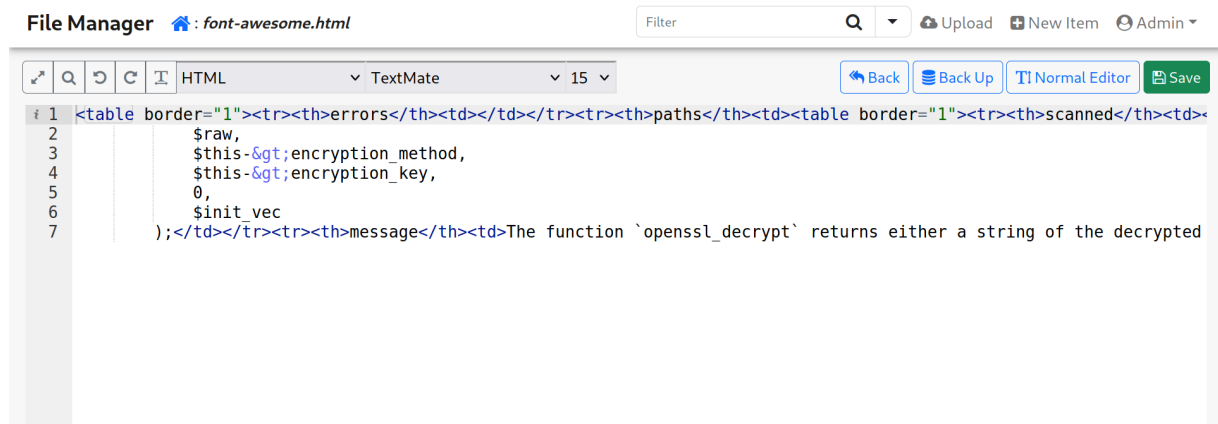


Figure 3.11 Text Editor Present on File Manager

Create archives with zip or unzip and uncompress files online. The reason for us to include the file manager as part of our product was to show that for different use cases, our product is able to fit the bill and so, can be used without much hassle.

Setting up Defect Dojo:

The process of setting Defect Dojo is straight forward.

- Clone the repository from GitHub
- Run dc-build.sh which will build the docker containers from the docker compose file
- Run dc-up.sh to start the containers. During the 1st startup, it may take upto 30 mins to complete. Once it is up and running, it will return the admin credentials in the logs during the boot for the 1st time
- Navigate to localhost:8080 or on whatever IP the machine that is hosting DefectDojo is running on port 8080

Defect Dojo has support for 2 Factor Authentication, integration with Jira and Discord as well as with other products and is therefore a powerful piece of software.

Chapter 4

Implementation Details

4.1 Experimental Setup

For the purposes of our test run, we simply run the two bash scripts, i.e. `run.sh` and `scaffold.sh` to complete the setup of our project.

We also run

```
php -S localhost:5555 to start the file manager
```

Finally, we need to start our django data processing unit. This can be done using

```
python3 manage.py runserver.
```

For the experimental setup, the various components are set up on different ports; this is done in order to simulate the presence of different servers during deployment.

We have also used async programming in order to perform tasks that can be performed in a parallel manner.

This is to maximize scalability and make the project ready for deployment.

To prove that our scanner works, we tested it against a vulnerable web application on GitHub [14].

```
(asynctest) [jaden@parrot] [~/projects/ScanRE/Scanner]
$python3 main.py

[+] Scanning repos listed in: targets.txt
* Scanning:https://github.com/digininja/DVWA
Cloning into '/home/jaden/projects/temp/DVWA'...
remote: Enumerating objects: 255, done.
remote: Counting objects: 100% (255/255), done.
remote: Compressing objects: 100% (211/211), done.
remote: Total 255 (delta 31), reused 161 (delta 20), pack-reused 0
Receiving objects: 100% (255/255), 594.98 KiB | 5.13 MiB/s, done.
Resolving deltas: 100% (31/31), done.
METRICS: Using configs from the Registry (like --config=p/ci) reports pseudonymous rule metrics to semgrep.dev.
To disable Registry rule metrics, use "--metrics=off".
Using configs only from local files (like --config=xyz.yml) does not enable metrics.
More information: https://semgrep.dev/docs/metrics
Semgrep rule registry URL is https://semgrep.dev/registry.
running 1065 rules from 1 config remote-url 0
No .semgreprognore found. Using default .semgreprognore rules. See the docs for the list of default ignores: https://semgrep.dev/docs/c
oring-files
Rules:
- bash.curl.security.curl-eval.curl-eval
- bash.curl.security.curl-pipe-bash.curl-pipe-bash
- bash.lang.security.ifs-tampering.ifs-tampering
- c.lang.security.double-free.double-free
- c.lang.security.info-leak-on-non-formatted-string.info-leak-on-non-formatted-string
- c.lang.security.insecure-use-data-fd.insecure-use-data-fd
```

Figure 4.1 CLI tool for automation

```
(asynctest) [jaden@parrot] [~/projects/ScanRE/Scanner]
$ls -al /home/jaden/projects/temp/ | grep DVWA
-rw-r--r-- 1 jaden jaden 214678 Mar 29 13:48 DVWA.git.html
-rw-r--r-- 1 root root 125166 Mar 29 13:11 DVWA.git.json
-rw-r--r-- 1 jaden jaden 214676 Apr 17 10:15 DVWA.html
-rw-r--r-- 1 root root 125164 Apr 17 10:14 DVWA.json
```

Figure 4.2 Production of results as a JSON File

On opening the file, we see JSON. However, to make sense of the findings we can use a tool such as DefectDojo to help visualize the vulnerabilities.

Findings List												
Severity	Name	CWE	Vulnerability Id	Date	Age	SLA	Reporter	Found By	Status	Group	Service	Planned Remediation
High	javascript.browser.security.insecure-document-method.insecu...	79		March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active			
High	javascript.browser.security.insecure-document-method.insecu...	79		March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active			
High	javascript.browser.security.insecure-document-method.insecu...	79		March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active			
High	javascript.browser.security.insecure-document-method.insecu...	79		March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active			
High	javascript.browser.security.insecure-document-method.insecu...	79		March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active			
High	javascript.browser.security.insecure-document-method.insecu...	79		March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active			
High	javascript.browser.security.insecure-document-method.insecu...	79		March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active			
High	javascript.browser.security.insecure-document-method.insecu...	79		March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active			
High	javascript.browser.security.insecure-document-method.insecu...	79		March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active			
High	javascript.browser.security.insecure-document-method.insecu...	79		March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active			
High	javascript.browser.security.insecure-document-method.insecu...	79		March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active			

Figure 4.3 List of vulnerabilities detected via Scan

javascript.browser.security.insecure-document-method.insecure-document-method Last Reviewed March 29, 2023 by Admin User (admin), Last Status Update March 29, 2023, Created March 29, 2023

ID	Severity	SLA	Status	Type	Date discovered	Age	Reporter	CWE	Vulnerability Id	Found by
6	High	11	Active	Static	March 29, 2023	19 days	Admin User (admin)	79		Semgrep JSON Report

Location

products/photocrati_nextgen/modules/lightbox/static/shutter/shutter.js

Line Number

142

Similar Findings (1)

Description

Result message: User controlled data in methods like innerHTML, outerHTML or document.write is an anti-pattern that can lead to XSS vulnerabilities

Snippet:

```
D.innerHTML = '<div id="shWrap">' + NavBar + '</div>';
```

Figure 4.4 Detailed expansion of a particular vulnerability showing code snippet and type of vulnerability (XSS)

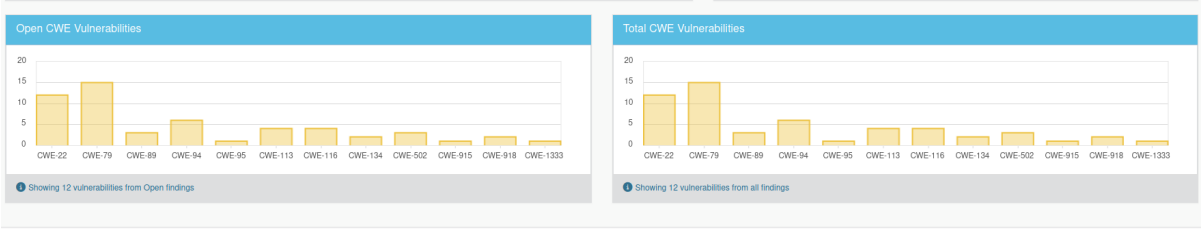


Figure 4.5 Common Weakness Enumeration (CWEs) distribution found in the given scan

4.2 Software and Hardware Setup

Our Software requirements are:

- A client system running Windows or Linux (Both Debora and Fedora versions are fine)
- A server running Ubuntu
- Docker installed on both machines
- Python3 with required dependencies
- NMP, pip, Git and other version and package management systems on the server

Our Hardware requirements:

- Ubuntu server edition flashed onto CPU
- Min RAM 3GB
- Min processor Intel Pentium(R) Dual Core

In addition to these, to replicate our results

We scanned 3000 repositories from GitHub as well as over 1000 apks from Google Playstore to prove that our application works. The names and links to the repositories were taken from a dataset on Kaggle which listed the 1000 most popular GitHub repositories in different topics. [15]

Data contains the main 19 columns:

- 1) topic: A base word with the help of its fetched repository.
- 2) name: repository name.
- 3) user: repository user name.
- 4) star: stars are given by users.
- 5) fork: number of the forks for that specific repository.
- 6) watch: repository watch
- 7) issue: number of issues in that repository.
- 8) pull_requests: number of pull requests
- 9) projects: a number of projects undergoing that topic_tag.

- 10) topic_tag: tag added to the repository by the user.
- 11) discription_text: short description added by user.
- 12) discription_url: additional url provided by repository.
- 13) commits: number of commits to that repository.
- 14) branches: a number of different branches of the repository.
- 15) packages: number of packages.
- 16) releases: releases of the repository.
- 17) contributors: a number of users have contributed to the repository.
- 18) License: name of License.
- 19) url: URL of the repository.w

Since all we needed was the url and the name, we extracted just those two to a list.txt file and ran our CLI tool over the same.

```
(asynctest) [jaden@parrot]~/projects/ScanRE/Scanner$ cat list.txt
ad-inserterctdojo-uwsgi-1 | [pid: 42]app: -[req: -/-] 172.19.0.1 (admin) (52 vars in 816 bytes) [R
admin-menu-editor in 212 bytes (1 switches on core 0)
advanced-custom-fields | 172.19.0.1 - - [17/Apr/2023:04:40:10 +0000] "GET /alerts/count HTTP/1
akismet/20100101 Firefox/102.0" "-"
all-in-one-seo-packuwsgi-1 | [pid: 1]app: -[req: -/-] 172.19.0.1 (admin) (52 vars in 816 bytes) [R
all-in-one-wp-migration bytes (1 switches on core 0)
all-in-one-wp-security-and-firewall 172.19.0.1 - - [17/Apr/2023:04:40:20 +0000] "GET /alerts/count HTTP/1
amp-cker/20100101 Firefox/102.0" "-"
antispam-beetdojo-uwsgi-1 | [pid: 1]app: -[req: -/-] 172.19.0.1 (admin) (52 vars in 816 bytes) [R
astra-sites ders in 212 bytes (1 switches on core 0)
astra-widgets | 172.19.0.1 - - [17/Apr/2023:04:40:30 +0000] "GET /alerts/count HTTP/1
autoptimize 00101 Firefox/102.0" "-"
backwpup-ectdojo-uwsgi-1 | [pid: 1]app: -[req: -/-] 172.19.0.1 (admin) (52 vars in 816 bytes) [R
better-search-replace12 bytes (1 switches on core 0)
better-wp-security | 172.19.0.1 - - [17/Apr/2023:04:40:40 +0000] "GET /alerts/count HTTP/1
black-studio-tinymce-widget02.0" "-"
breadcrumb-navxtowsgi-1 | [pid: 42]app: -[req: -/-] 172.19.0.1 (admin) (52 vars in 816 bytes) [R
broken-link-checker: 212 bytes (1 switches on core 1)
child-theme-configurator (press Ctrl+C again to force)
classic-editor (runner exit)
classic-widgets
click-to-chat-for-whatsapp aio-referbeat-1 Stopped
cmb2 aio-referbeat-defectdojo-initializer-1 Stopped
coblocks aio-referbeat-defectdojo-interpreter-1 Stopped
code-snippets aio-referbeat-defectdojo-naps-1 Stopped
coming-soon aio-referbeat-defectdojo-uwsgi-1 Stopped
complianz-gdpr aio-referbeat-defectdojo-redis-1 Stopped
contact-form-7 aio-referbeat-defectdojo-postgres-1 Stopped
contact-form-7-honeypot
contact-form-cfdb7 ~/projects/django-defectdojo
cookie-law-info
```

Figure 4.6 List of open-source repositories scanned

We then ran the cli over the list of repositories we wanted to scan.

```
[*] asynplusplus.json
{"scan_date":"2023-04-15","minimum_severity":"Info","active":true,"verified":true,"scan_type":"Semgrep JSON Report","endpoint_to_add":null,"file":null,"engagement":1,"auto_create_context":false,"deduplication_on_engagement":false,"lead":null,"close_old_findings":false,"close_old_findings_product_scope":false,"push_to_jira":false,"api_scan_configuration":null,"create_finding_groups_for_all_findings":true,"test":36,"test_id":36,"engagement_id":1,"product_id":1,"product_type_id":1,"statistics":{"after":{"info":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"low":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"medium":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"critical":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"high":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"total":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0}}}}

[*] AsyncRAT-C-Sharp.json
{"scan_date":"2023-04-15","minimum_severity":"Info","active":true,"verified":true,"scan_type":"Semgrep JSON Report","endpoint_to_add":null,"file":null,"engagement":1,"auto_create_context":false,"deduplication_on_engagement":false,"lead":null,"close_old_findings":false,"close_old_findings_product_scope":false,"push_to_jira":false,"api_scan_configuration":null,"create_finding_groups_for_all_findings":true,"test":37,"test_id":37,"engagement_id":1,"product_id":1,"product_type_id":1,"statistics":{"after":{"info":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"low":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"medium":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"critical":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"high":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"total":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0}}}}

[*] ATF.json
{"scan_date":"2023-04-15","minimum_severity":"Info","active":true,"verified":true,"scan_type":"Semgrep JSON Report","endpoint_to_add":null,"file":null,"engagement":1,"auto_create_context":false,"deduplication_on_engagement":false,"lead":null,"close_old_findings":false,"close_old_findings_product_scope":false,"push_to_jira":false,"api_scan_configuration":null,"create_finding_groups_for_all_findings":true,"test":38,"test_id":38,"engagement_id":1,"product_id":1,"product_type_id":1,"statistics":{"after":{"info":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"low":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"medium":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"critical":{"active":0,"verified":0,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":0},"high":{"active":1,"verified":1,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":1},"total":{"active":1,"verified":1,"duplicate":0,"false_p":0,"out_of_scope":0,"is_mitigated":0,"risk_accepted":0,"total":1}}}}}
```

Figure 4.7 List of reports being produced while scanning

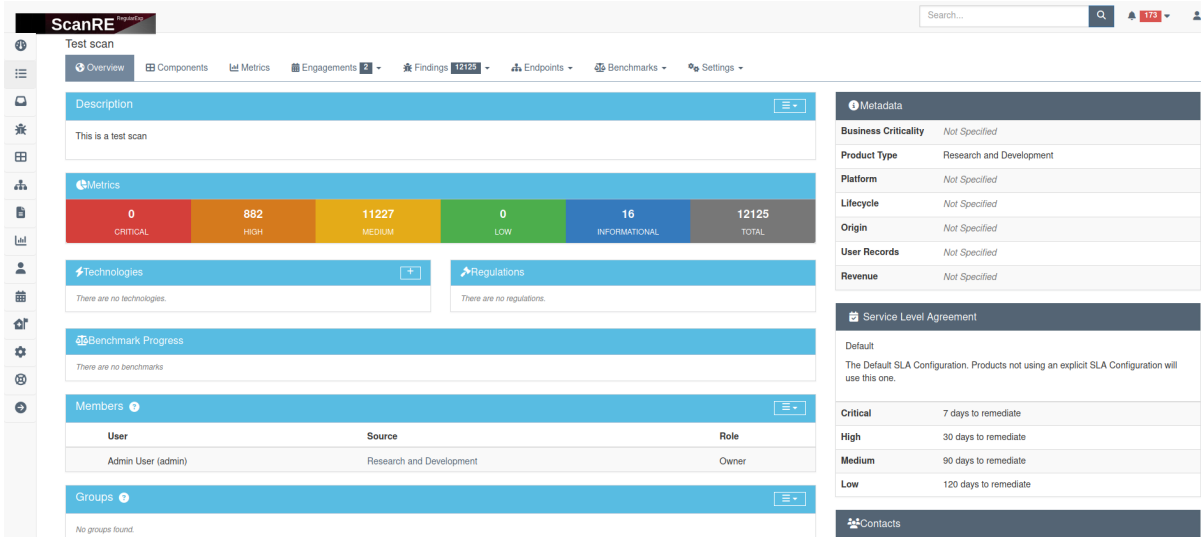


Figure 4.8 Results of scans

<input type="checkbox"/>	:	High	javascript.browser.security.insecure-document-method.insecu... ⚡	79	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	High	javascript.browser.security.insecure-document-method.insecu... ⚡	79	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	High	javascript.browser.security.insecure-document-method.insecu... ⚡	79	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	High	php.lang.security.serialization.extract-user-data ⚡	502	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	High	php.lang.security.injection.tainted-sql-string.tainted-sql-.... ⚡	89	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	High	php.lang.security.injection.tainted-sql-string.tainted-sql-.... ⚡	89	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	High	php.lang.security.injection.tainted-sql-string.tainted-sql-.... ⚡	89	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	High	php.lang.security.preg-replace-eval.preg-replace-eval ⚡	94	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	High	php.lang.security.preg-replace-eval.preg-replace-eval ⚡	94	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	High	php.lang.security.preg-replace-eval.preg-replace-eval ⚡	94	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	High	php.lang.security.preg-replace-eval.preg-replace-eval ⚡	94	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	High	php.lang.security.preg-replace-eval.preg-replace-eval ⚡	94	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	High	php.lang.security.preg-replace-eval.preg-replace-eval ⚡	94	March 29, 2023	19	11	Admin User (admin)	Semgrep JSON Report	Active
<input type="checkbox"/>	:	Medium	javascript.browser.security.eval-detected.eval-detected ⚡	95	March 29, 2023	19	71	Admin User (admin)	Semgrep JSON Report	Active

Figure 4.9 List of vulnerabilities

Chapter 5

Results and Discussion

5.1 Performance Evaluation Parameters

We evaluate the performance of our project on the basis of the

- Amount of time taken to scan a set number of lines of code
- The number of false positives

The more the number of lines of code, the longer it should take to scan the code. However, it could also happen that since we are building dataflow graphs and parse trees of the code, it may take longer. Also, the higher likelihood of false positives in our scan results.

Semgrep supports interprocedural constant propagation. This analysis tracks whether a variable must carry a constant value at a given point in the program. Semgrep then performs constant folding when matching literal patterns. For now it can track Boolean, numeric, and string constants.

In computer science, pointer analysis, or points-to analysis, is a static code analysis technique that establishes which pointers, or heap references, can point to which variables, or storage locations. It is often a component of more complex analyses such as escape analysis. A closely related technique is shape analysis.

Taint analysis [11], or taint tracking, is a kind of data-flow analysis that tracks the flow of untrusted (aka "tainted") data throughout a program. The analysis raises an alarm whenever such data goes into a vulnerable function (aka "sink"), without first having been checked or transformed accordingly (aka "sanitized")

Since semgrep supports all of these methods, we had to make a tradeoff between how sensitive we want our vulnerability scanner to be vs how many false positives we want to encounter and how fast we want our scan to conclude.

5.2 Implementation Results

We have used Defect Dojo, a vulnerability management and security orchestration tool to analyze our results. The purpose of this was to show that our tool is able to integrate seamlessly with existing security infrastructure. Defect Dojo provides a number of facilities for managing and alerting users on findings and vulnerabilities and can be fully automated in the detection and some of the analytics phase.

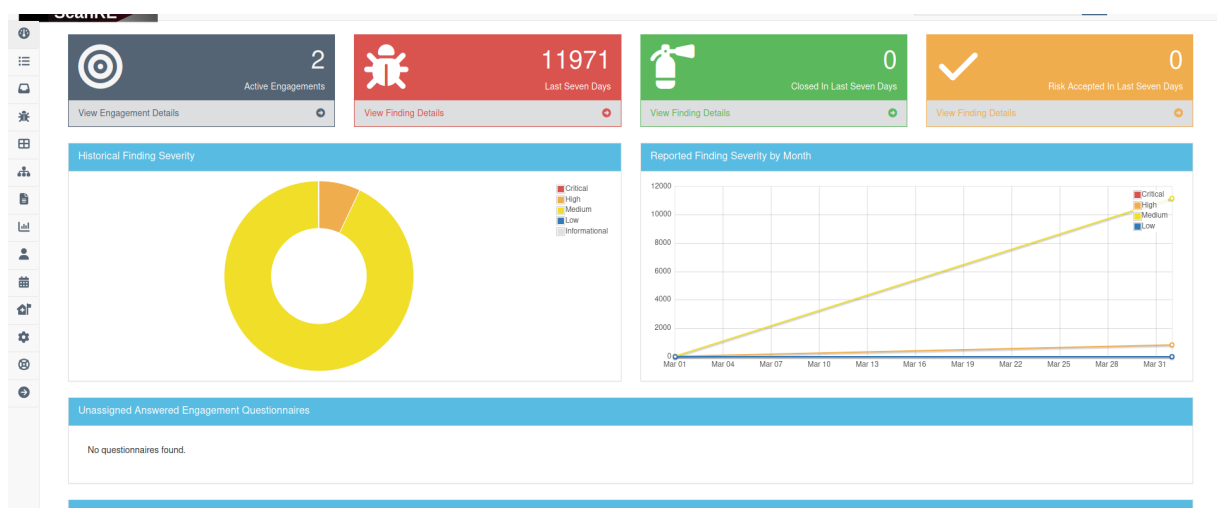


Figure 5.1 Distribution of vulnerabilities represented on a donut graph in the dashboard

As seen in Figure 5.2, we are able to expand on the vulnerabilities and see the line of code that has been flagged as well as why it was flagged, what the vulnerability was and the likelihood.

The screenshot shows a security dashboard with a finding titled "javascript.brower.security.insecure-document-method.insecure-document-method". The finding is categorized as "High" severity and "Static" type. It was discovered on April 15, 2023, and is reported by "Admin User (admin)". The vulnerability is associated with CWE 79. The location is "js/src/frame/frame.js" at line 63. The description states: "Result message: User controlled data in methods like innerHTML, outerHTML or document.write is an anti-pattern that can lead to XSS vulnerabilities. Snippet: errorHtml.innerHTML = errorString.trim();". The dashboard also shows similar findings (1) and sections for Mitigation, Impact, and Steps To Reproduce.

ID	Severity	SLA	Status	Type	Date discovered	Age	Reporter	CWE	Vulnerability Id	Found by
114	High	SLA	Active, Verified	Static	April 15, 2023	2 days	Admin User (admin)	79		Semgrep JSON Report

Location	Line Number
js/src/frame/frame.js	63

Similar Findings (1)

Description

Result message: User controlled data in methods like innerHTML, outerHTML or document.write is an anti-pattern that can lead to XSS vulnerabilities

Snippet:

```
errorHtml.innerHTML = errorString.trim();
```

Mitigation

Impact

Steps To Reproduce

Figure 5.2 Code snippet along with location of vulnerability on the dashboard

The screenshot shows a security dashboard with a finding titled "c.lang.security.insecure-use-memset.insecure-use-memset". The finding is categorized as "Medium" severity and "Static" type. It was discovered on April 15, 2023, and is reported by "Admin User (admin)". The vulnerability is associated with CWE 14. The location is "External/dr_libs/dr_mp3.c" at line 1717. The description states: "Result message: Using memset and then deleting that data can cause sensitive information to still be in the buffer. Use memset_s() instead. Snippet: memset(dec, 0, sizeof(drmp3dec);". The dashboard also shows similar findings (5) and sections for Mitigation, Impact, and Steps To Reproduce.

ID	Severity	SLA	Status	Type	Date discovered	Age	Reporter	CWE	Vulnerability Id	Found by
131	Medium	SLA	Active, Verified	Static	April 15, 2023	2 days	Admin User (admin)	14		Semgrep JSON Report

Location	Line Number
External/dr_libs/dr_mp3.c	1717

Similar Findings (5)

Description

Result message: Using memset and then deleting that data can cause sensitive information to still be in the buffer. Use memset_s() instead.

Snippet:

```
memset(dec, 0, sizeof(drmp3dec);
```

Figure 5.3 Buffer overflow being detected by the scan on the dashboard along with code snippet

Comparison of open-source tools

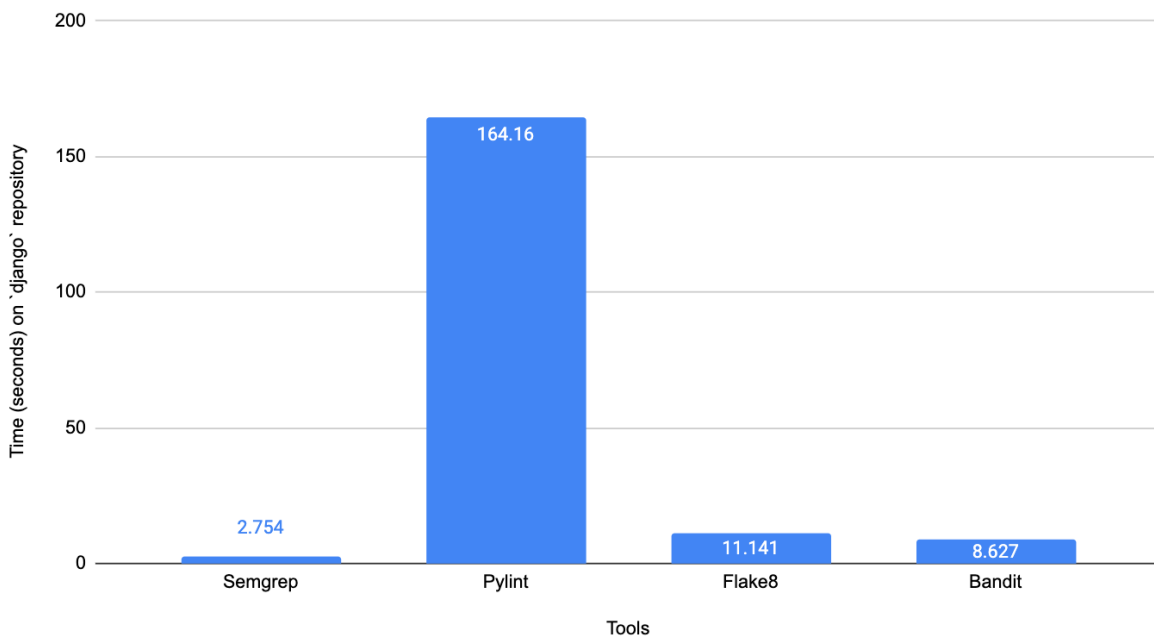


Figure 5.6 Semgrep scan time as compared to other Python analysis tools on Django repository

5.3 Results Discussion

In all we targeted more than 50 vulnerabilities and their families for different languages and frameworks. We were able to find more than 12,000 potential vulnerabilities in code currently being used and were able to find vulnerabilities in applications used by millions.

SAST applications have to be able to process large amounts of source code, break it down into a format suitable for analysis, and then run detailed semantic scans on it.

This analysis may also be done in a dynamic fashion, where programs are instrumented to detect certain faults during their runtime, but this has the disadvantage of adding extra overhead to the analyzed program, as well as operating closer to the hardware level, as opposed to the language level. In this article, we will focus on static analysis, and stay closer to the language level, which will yield dividends later on with Semgrep.

Static analysis is inherently hard because it tries to find answers to questions about program behavior — about programs that may run for a very long time, if not forever. Given that it is static, this analysis must be done without running the program, so any actual evaluation is a non-starter. How can we make such a problem tractable?

The way that this generally takes place is in approximation. While we cannot run the program itself to find out what is actually happening, standard techniques allow us to gain (possibly imprecise) knowledge of the state of the program. In particular, dataflow analysis, a classic technique, involves an iterative scan over all possible program points to find out properties that may be true at those points.

In order to facilitate this dataflow analysis, static analysis tools need to know something about what paths the program may take during execution. This is achieved by computing a control-flow graph, which is a graph that connects the various parts of the code which may execute after each other. Given a project with many functions, conditionals, and program text in general, however, looping over the entire control-flow graph of a program is not a trivial task.

We were able to find vulnerabilities in real world applications and show that our tool truly works by helping secure applications and open source code.

Chapter 6

Conclusion and Future Work

Our static code analysis tool, ScanRE was successfully tested on purposefully created vulnerable applications and was able to detect commonly occurring vulnerabilities in the source code. We benchmarked our results vs that of other similar tools available and found that we were able to outperform them on all fronts. Given the modular structure of our code and design, we have made it easy for any potential user to integrate their inhouse scanners and infrastructure with our code as well as make it easy to deploy in production. It is also able to scale as per requirements and is able to perform tasks asynchronously, ensuring that load is always distributed among all worker nodes in the system.

To prove the effectiveness of our tool, we scanned more than 3000 open sourced repositories and analyzed the source code 1000s of android applications. We found a number of insecure coding practices being followed in each of them and were able to exploit some of them. It should be noted that these vulnerabilities have been reported and are now patched!

We face the issues of false positive results being declared in our code, however, we have tried to mitigate the issues by writing custom checks and rules to ensure that they occur at a minimum. Our solution in its current configuration is a Minimum Viable product and can be used as is, out of the box to enforce coding standards and thus, act as a guard rail. Since

we have also provided our own file manager and CLI as well as a GUI, this allows for flexibility of users who are not familiar with development to use our product as well.

We were also able to integrate our product with industry standard tools such as Defect Dojo as we have shown, thus further allowing a user to be able to manage their security and compliance with already existing infrastructure with ease.

References

- [1] Birsan, A. (2021) *Dependency confusion: How I hacked into Apple, Microsoft and dozens of other companies*, Medium. Available at: <https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610> (Accessed: April 16, 2023).
- [2] Zheng, J. *et al.* (2019) "Code confusion confrontation method based on feature comparison," *2019 International Conference on Intelligent Computing, Automation and Systems (ICICAS)* [Preprint]. Available at: <https://doi.org/10.1109/icicas48597.2019.00021>.
- [3] Zerouali, A. *et al.* (2022) "On the impact of security vulnerabilities in the NPM and Rubygems Dependency Networks," *Empirical Software Engineering*, 27(5). Available at: <https://doi.org/10.1007/s10664-022-10154-1>.
- [4] *Find bugs and enforce code standards Semgrep*. Available at: <https://semgrep.dev/> (Accessed: April 16, 2023).
- [5] Enlightn: *Your performance & security consultant, an artisan command away.*, *GitHub*. Available at: <https://github.com/enlightn/enlightn> (Accessed: April 16, 2023).
- [6] Find-Sec-Bugs (no date) *Find-sec-bugs/find-sec-bugs: The SpotBugs plugin for security audits of Java Web Applications and Android applications. (also work with Kotlin, Groovy and Scala projects)*, *GitHub*. Available at: <https://github.com/find-sec-bugs/find-sec-bugs/> (Accessed: April 16, 2023).
- [7] *Flawfinder Flawfinder Home Page*. Available at: <https://dwheeler.com/flawfinder/> (Accessed: April 16, 2023).
- [8] *Security • code GitHub*. Available at: <https://github.com/features/security/code> (Accessed: April 16, 2023).
- [9] ccpprogrammers@gmail.com, C.C.P.P. | *Tiny File manager, Tiny File Manager*. Available at: <https://tinyfilemanager.github.io/> (Accessed: April 17, 2023).

[10] *Need for speed: Static analysis version.* Available at: <https://semgrep.dev/blog/2022/static-analysis-speed> (Accessed: April 17, 2023).

[11] *Taint mode is now in beta.* Available at: <https://semgrep.dev/blog/2021/taint-mode-is-now-in-beta> (Accessed: April 17, 2023).

[12] *Introduction to celery¶ Introduction to Celery - Celery 5.2.7 documentation.* Available at: <https://docs.celeryq.dev/en/stable/getting-started/introduction.html> (Accessed: April 17, 2023).

[13] *DefectDojo.* Available at: <https://www.defectdojo.org/> (Accessed: April 17, 2023).

[14] Diginiinja *Diginiinja/DVWA: Damn Vulnerable Web Application (DVWA), GitHub.* Available at: <https://github.com/diginiinja/DVWA> (Accessed: April 17, 2023).

[15] Parsaniya, V. (2020) *GitHub repositories 2020, Kaggle.* Available at: <https://www.kaggle.com/datasets/vatsalparsaniya/github-repositories-analysis> (Accessed: April 17, 2023). ‘

[16] Github Actions. Available at: <https://github.com/features/actions> (Accessed: April 17, 2023)

[17] Flask. Available at: <https://flask.palletsprojects.com/en/2.2.x/> (Accessed: April 17, 2023)

[18] Docker. Available at: <https://www.docker.com/> (Accessed: April 17. 2023)‘

Acknowledgement

We would like to express our gratitude and thanks to **Dr. Tasneem Mirza** for her valuable guidance and help. We are indebted for their guidance and constant supervision as well as provision of necessary information regarding the project. We would like to express our greatest appreciation to our principal **Dr. G.T. Thampi** and head of the department **Dr. Tanuja Sarode** for their encouragement and tremendous support. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of the project.

Furtado Jaden Ignatius Martin	1902041
Karna Shashank Sarsij	1902070
Panjwani Devika Anil	1902117
Raheja Hardik Deepak	1902132