Enhancing Container Security Orchestration and Management Tool

MSc Research Project
Cybersecurity

Shivam Rajesh Tiwari
Student ID: 22102396

School of Computing
National College of Ireland

Supervisor: Jawad Salahuddin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shivam Rajesh Tiwari		
Student ID:	22102396		
Programme:	MSc Cybersecurity	**Year:**	2023
Module:	Academic Internship		
Supervisor:	Jawad Salahuddin		
Submission Due Date:	14-12-2023		
Project Title:	Enhancing Container Security Orchestration and Management Tool		
Word Count:	5280 **Page Count** 21		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Shivam Rajesh Tiwari

Date: 14-12-2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Enhancing Container Security Orchestration and Management Tool

Shivam Rajesh Tiwari
22102396

Abstract

With containerization's rapid growth, security lags as a pressing issue. This research develops automated methodologies to enhance protection for container workloads. Containers' dynamic nature necessitates flexible security able to address evolving threats. The project delivers a parallel scanning pipeline tightly coupled with CI/CD automation to enable enhanced security. Implementing multithreading reduces scan times by 40% versus sequential approaches while preserving accuracy. Seamless integration with developer workflows proactively uncovers vulnerabilities early in the lifecycle. Initial testing shows promise for the methodology securing cloud-native applications. Further enhancements and expanded compatibility are beneficial future work. By combining performance and scalability with organizational fit, this research pushes the leading edge of innovation for container security, laying groundwork for widespread impact across industries.

1 Introduction

With its unmatched flexibility and efficiency, containerization has become a revolutionary force in the fast-paced world of modern software development. DevOps approaches now revolve around technologies like Docker and Kubernetes, which let developers build, ship, and execute apps consistently across a variety of settings. But the swift adoption of containerization also presents a complicated set of issues, the most pressing of which being security. Organizations must strengthen their security protocols at the same time that they work to fully realize the advantages of containers. In order to better address the changing threat landscape, this research paper explores container security in great detail and aims to improve security orchestration and management solutions now in use.

1.1 Context and Importance

Containerization has become more popular due to application developers' constant quest for scalability and agility. Applications and their dependencies are encapsulated by containers, which create a reproducible and isolated environment. This promotes uniformity throughout the stages of development and deployment, but it also raises security issues that require careful consideration and resolution. The paper lays the groundwork by exploring the complexities of container security and recognizing the need to carefully combine innovation with strong security measures. This research is important because it is dedicated to improving container security without sacrificing the speed and flexibility that containers are meant to offer.

1.2 The Situation of Container Security Tools Right Now

A thorough analysis of the state of container security technologies today finds a varied ecosystem that addresses different aspects of container security. While runtime security is handled by programmes like Falco, image vulnerability evaluations are the main focus of tools like Docker Security Scanning. However, security tactics must constantly change due to the dynamic nature of containerized systems. The basis for suggestions that go beyond small tweaks to achieve a comprehensive and flexible security system is laid by this examination.

1.3 The Research's Goals

The study presents a complete set of goals, each designed to strengthen container security:

Vulnerability Detection and Remediation: The main goal of the project is to develop methods for locating vulnerabilities in container images during build time. It also looks at innovative method of simultaneous scanning for effective vulnerability detection, reducing time and resources used to scan the containers.

Optimized Resource Utilization: The secondary goal of the project is to optimize the resource utilization which includes computation and manpower required. The project aims to cut down the resource utilization.

Integration with DevOps Pipelines: The article recognizes the difficulties in incorporating improved security features into DevOps pipelines in a smooth and efficient manner. In order to guarantee that security becomes an essential component of continuous integration and deployment processes, it aims to provide scalable and effective integration.

1.4 Originality

The research is expected to make significant contributions to the forefront of container security.

Reduced time for scanning: The time for scanning multiple containers should be drastically reduced, after the solution is built up. Additionally, resources i.e., manpower required to setup all the scans are drastically reduced.

Reducing the resource requirement: When it comes to resources, in the current scenario the tools multiple instances should be created to scan containers, leading to use of many resources, which is not an optimistic approach and leads to waste of resource. Also, there is a licensing factor that cannot be neglected. With the solution, all these problems should be solved, and only one resource is needed and based on the number of container scan, computational power can be adjusted.

Integrating Scalable with CI/CD Pipelines: The study proposes a seamless integration of upgraded security features, acknowledging the critical role that continuous integration/delivery chain (CI/CD) pipelines play in modern software delivery. Because of the scalability of this connection, security will be an efficient and fundamental part of agile DevOps processes.

2 Related Work

(Mavridis et al., 2023) make contributions to the field through their investigation of virtual machines, unikernels, and orchestrated sandboxed containers. With an emphasis on improving isolation in serverless computing and multitenant workloads, the paper explores cutting edge methods for protecting and maximising cloud-based apps. By contrasting various virtualization technologies, the study contributes to the comprehension of the advantages and disadvantages of each, providing insightful information that is beneficial to both scholars and practitioners. In their study, (Čilić et al., 2023) discuss how container orchestration solutions function in edge computing scenarios. The report offers a thorough review of different orchestration technologies and sheds light on their efficacy in edge circumstances through a methodical evaluation. For stakeholders looking to deploy containers at the edge, this research is essential since it provides recommendations for the best orchestration solution based on needs and performance data. The PhD dissertation of (Alqarni et al., 2023) makes a major contribution to cloud security and privacy. The study provides improvements through vulnerability assessment and zero-knowledge encryption, with a focus on Kubernetes deployments. This paper provides workable ways to protect sensitive data in cloud-based applications by tackling important security issues in containerised settings. The results add to the current endeavours to strengthen security protocols in Kubernetes installations. In their study, (Moreau et al., 2023) investigate how containers support computational repeatability. Their research highlights how crucial containers are to making computer experiments easier to replicate. This work adds significant insights to the scientific community by offering a thorough review of container usage for reproducibility, particularly in fields where experiment reproducibility is crucial. Future directions of cloud-native workload orchestration at the edge are presented by (Vaño et al., 2023) along with a deployment review. The complexity of coordinating workloads in edge computing environments is examined in this paper. This study is essential to comprehending the special difficulties presented by edge environments and offers recommendations for managing containerised workloads in dispersed and resource-constrained contexts.

A Docker Container Security Orchestration and Posture Management Tool is presented by Perera, Reza, De Silva, Karunarathne, Ganegoda, and Senarathne (2022). Docker container security orchestration is a critical need that this research, presented at the 13th International Conference on Computing Communication and Networking Technologies, aims to solve. By improving the overall security posture of Dockerized apps, the tool hopes to advance container security as it develops. (Raj et al., 2018) examine the complexities of operating containerised apps across several cloud environments as they go into automated multi-cloud operations and container orchestration. Their research, which was published in "Software-Defined Cloud Centres," offers insightful information about the management and operational tools required to orchestrate containers in a multi-cloud environment. The field is enhanced by (Hoque et al., 2017) as they examine the opportunities and difficulties related to container orchestration in fog computing infrastructures. This work offers the foundation for understanding how container orchestration might be extended to fog computing environments, taking into account their particular characteristics and requirements. It was presented at the IEEE 41st Annual Computer Software and Applications Conference. In this paper, (Sultan et al., 2019) discuss the problems, difficulties, and potential future directions in the field of container security, offering a thorough overview of the subject. The report, which was published in IEEE Access, is a useful tool for comprehending the container security landscape as a whole because it highlights the dangers that are always changing as well as possible countermeasures. (MP et al., 2016) concentrate on using Linux hardening strategies to improve the security of Docker containers. Their research, which was presented at the 2nd International Conference on Applied and Theoretical Computing and Communication Technology, explores methods with roots in Linux hardening practices and adds to the ongoing efforts to strengthen Docker security.

The autonomic orchestration of containers is examined by (Casalicchio et al., 2016), who also describes the problem definition and research challenges related to this area. The article, which was presented at VALUETOOLS, explores the difficulties in attaining autonomous behaviour in container orchestration, highlighting important problems and outlining areas in need of more research. By suggesting automated vulnerability scanning and repair using (Trivy et al., 2023) advances container security. The publication's emphasis on automated security procedures implies a proactive strategy to finding and addressing vulnerabilities within containerised environments, even though precise details are not disclosed. The notion of Container Security Intelligence is presented by (Mahavaishnavi et al., 2023), who use machine learning to detect anomalies in containerised systems. The research, which was published in the Journal of Propulsion Technology, adds to the developing topic of container security by examining how machine learning may be used to improve security by identifying anomalies within containerised settings. With Lic-Sec, an improved AppArmor Docker security profile generator, (Zhu et al., 2021) advance container security. The study, which was published in the Journal of Information Security and Applications, presents a tool that focuses on using the AppArmor architecture to generate enhanced security profiles for Docker containers, improving the overall security posture of containerised applications. The study conducted by (Sharma et al., 2016) compares virtual machines and containers on a large scale. This study, which was presented at the 17th International Middleware Conference, explores the scalability and performance of virtual machines and containers, offering important insights into each technology's potential applications Docker's suitability for use in industrial IoT gateways is evaluated by Lumio (2018). The study investigates the useful applications of containerisation technology, Docker, in industrial Internet of things environments. The paper, which was published in an undisclosed source, probably offers advice on how to use Docker to improve the efficiency and flexibility of industrial IoT gateways.

A survey of cloud container technologies that is up to date is provided by (Pahl et al., 2017). The study, which was published in the IEEE Transactions on Cloud Computing, gives a thorough picture of the state of the industry by summarising the developments, difficulties, and current trends in cloud container technologies. (Senel et al., 2023) offers a doctoral dissertation from Sorbonne Université on container orchestration for edge clouds. The dissertation probably delves into the complexities of managing containers in edge computing settings, illuminating the unique problems and approaches associated with container deployment at the network's edge. (VS et al., 2023) explore container security, discussing levels of precaution, mitigating techniques, and offering research perspectives in their publication published in Computers & Security. This document probably provides a thorough analysis of security protocols in the container environment, along with risk-reduction tactics and recommendations for future research topics. An investigation of the Docker ecosystem's vulnerabilities is carried out by Martin et al., 2018) and is published in Computer Communications. In order to improve the overall security of Docker-based applications, the paper probably analyses potential vulnerabilities within the Docker ecosystem and offers insights into areas that might need to be strengthened. In an arXiv preprint, (Bui et al., 2015) analyses Docker security. The study most likely examines Docker's security features, providing information on possible weaknesses, advantages, and areas where the Docker framework needs to be strengthened. A survey on multi-access edge computing security and privacy is presented by (Ranaweera et al., 2021) in the IEEE Communications Surveys & Tutorials. This study probably gives a thorough overview of privacy and security issues in multi-access edge computing, highlighting problems and possible fixes. (Zhu et al., 2023) have published an article in SN Computer Science regarding the creation of access security policies for cloud services that utilise containers. It is probable that the article will examine techniques and approaches for creating efficient access control policies that are especially suited for cloud-deployed containerised applications.

(Jain et al., 2021) make a significant addition to the field of container security by their investigation of Docker image static vulnerabilities. They stress in their work how critical it is to recognise and address security threats in containerised environments. The authors want to improve the overall security posture by proactively identifying vulnerabilities in Docker images through the use of static analysis techniques. The paper tackles the crucial topic of containerised application security, offering perceptions on possible risks and weaknesses that can be useful in creating strong security procedures for Docker-based implementations. By concentrating on the evaluation of container security vulnerability detection methods, (Javed et al., 2021) make a contribution to the field of container security. In order to determine the effectiveness and calibre of current techniques for identifying vulnerabilities in containerised environments, their task entails a thorough investigation. The authors offer insightful information on the state of container security solutions by analysing the benefits and drawbacks of various tools. In order to improve overall system security and resilience, practitioners and organisations seeking to make well-informed decisions on the choice and application of security tools in their containerised infrastructure will find this research to be crucial. The "UBCIS: Ultimate Benchmark for Container Image Scanning" is presented by (Berkovich et al., 2020) as a CSET at the USENIX Security Symposium. The study presents UBCIS, a benchmark intended to assess container image scanning methods in a thorough manner. The authors hope to offer a standardised framework for evaluating the efficacy and efficiency of different container image scanning technologies by creating this benchmark. By addressing the requirement for a consistent evaluation approach, the research advances the field of cybersecurity by empowering practitioners and academics to make well-informed choices when evaluating and choosing container image scanning solutions in various security situations. In their dissertation, (Andersson et al., 2022) examine issues with Docker container images and the existing tools for scanning them. The paper explores the security features of Docker container images, addressing concerns about current image security procedures and scanning technologies in particular. Through examining these issues, the study offers important new perspectives on the difficulties and possible weaknesses related to containerisation. The dissertation lays the groundwork for future developments in container security procedures by offering a thorough resource for comprehending and improving the security environment of Docker container images.

3 Research Methodology

For the purpose of this research project, we executed multiple Docker images in parallel to find vulnerabilities. This was accomplished by implementing a multithreading technique in Python that allowed for the simultaneous scanning of multiple Docker images while dividing the workload among the threads.

3.1 Tool Selection

There are many static vulnerability scanning tools available when it comes to container scanning. Some of the most popular once are Claire, Anchore, Trivy, Dagda, Falco, Qualys, Grype, etc. Majority of the tools are open source and can be used without licence requirements. Although for the project requirements, Claire, Anchore and Trivy suits best. Following is a comparison table, for the above-mentioned tools.

Feature	**Trivy**	**Anchore**	**Clair**
Ease of Use	Easy to use with a user-friendly interface	User-friendly, but may have more features	Lightweight but might have fewer features
Scanning Speed	Known for fast scanning capabilities	Scanning speed is generally efficient	Lightweight but may be slower in some cases
Vulnerability Types	CVEs in OS packages, applications, IaC	OS packages, libraries, application deps	Focuses on OS vulnerabilities in images
Integration	Easily integrated into CI/CD pipelines	Can be integrated into CI/CD workflows	Integration capabilities may vary
Cost	Free and open-source	Open-source with a commercial offering	Open-source and free to use
Deployment Options	Helm, Docker, ECR	Docker, Kubernetes, other deployment ops	Typically used with Docker, Kubernetes

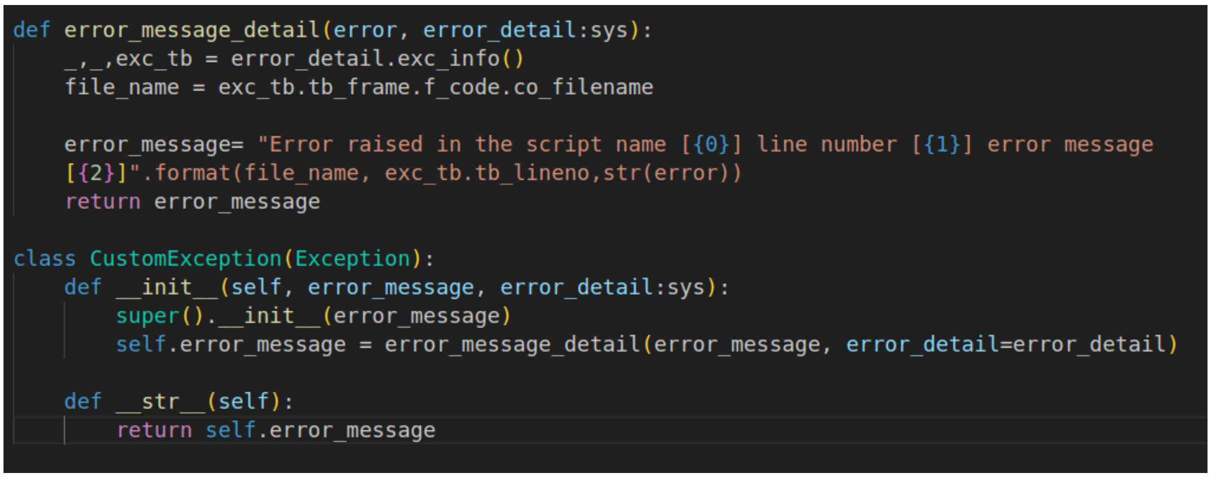
Community Support	Active community support	Active community, commercial support	Open-source project with community support
IaC Scanning Support	Supports IaC in Terraform files	Comprehensive IaC scanning support	Focuses more on Docker image scanning, limited IaC support
Custom Policies	Allows custom policies	Supports customizable policies for scan	Policies are predefined, limited customization
Notification Alerts	Supports notification alerts for vulns	Notifications for policy violations	Basic notification system for vulns
License	Open-source	Open-source with commercial offering	Open-source

Table 1 : Comparison Table of Container Scanning Tools

Based on the requirements and features, Trivy was selected as the tool, due to its versatility, speed, ease of use and comprehensive scanning capabilities. After, going through many research papers on Container scanning tools, Trivy was one the best choices.

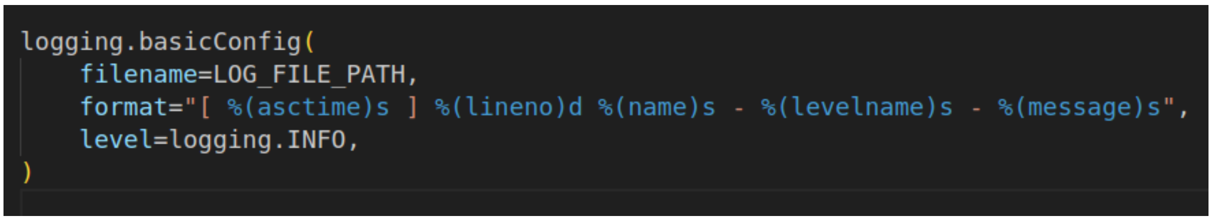
3.2 Setting up the Logging and Exception Handling Modules:

For efficient exception handling and logging, implementing and testing the logger.py and exception.py modules were done. Unexpected occurrences in the code are handled via exception.py. This means to create exceptions based on the requirements and recording the data. It can specify destinations (such as a file or terminal), log levels, and formats. It's essential for debugging and following the application's flow.



```python
def error_message_detail(error, error_detail:sys):
    _,_,exc_tb = error_detail.exc_info()
    file_name = exc_tb.tb_frame.f_code.co_filename

    error_message= "Error raised in the script name [{0}] line number [{1}] error message
    [{2}]".format(file_name, exc_tb.tb_lineno,str(error))
    return error_message

class CustomException(Exception):
    def __init__(self, error_message, error_detail:sys):
        super().__init__(error_message)
        self.error_message = error_message_detail(error_message, error_detail=error_detail)

    def __str__(self):
        return self.error_message
```

Figure 1: Error Handling

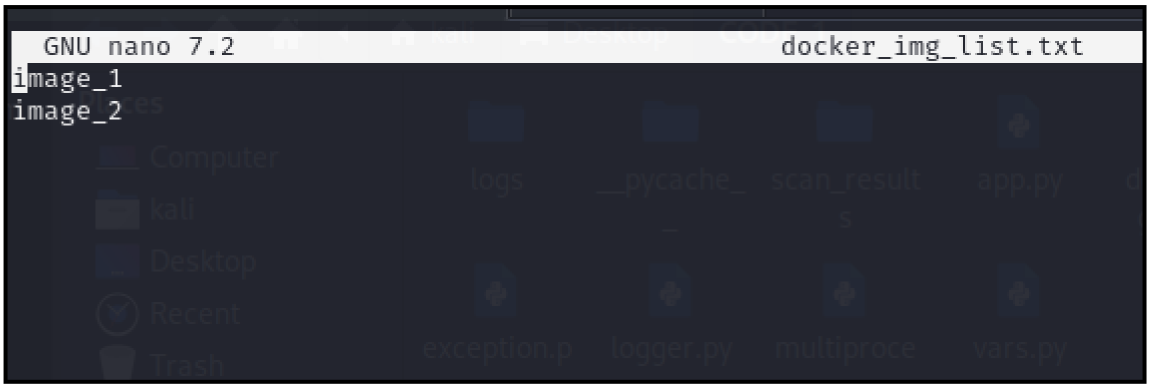


```python
logging.basicConfig(
    filename=LOG_FILE_PATH,
    format="[ %(asctime)s ] %(lineno)d %(name)s - %(levelname)s - %(message)s",
    level=logging.INFO,
)
```

Figure 2: Logging

3.3 Creating a file that contains all the images:

It is important that the image names in the Docker_img_list.txt file are accurate and the image exists, to ensure the scanner to work properly. Implemented error handling to address formatting and file existence problems. A list of Docker images to be scanned is stored in the Docker_img_list.txt file. Each line should contain only one docker image name. Included checks in the code to ensure that the file exists and is formatted correctly. In the event that a file is missing or improperly formatted, error handling has been placed.

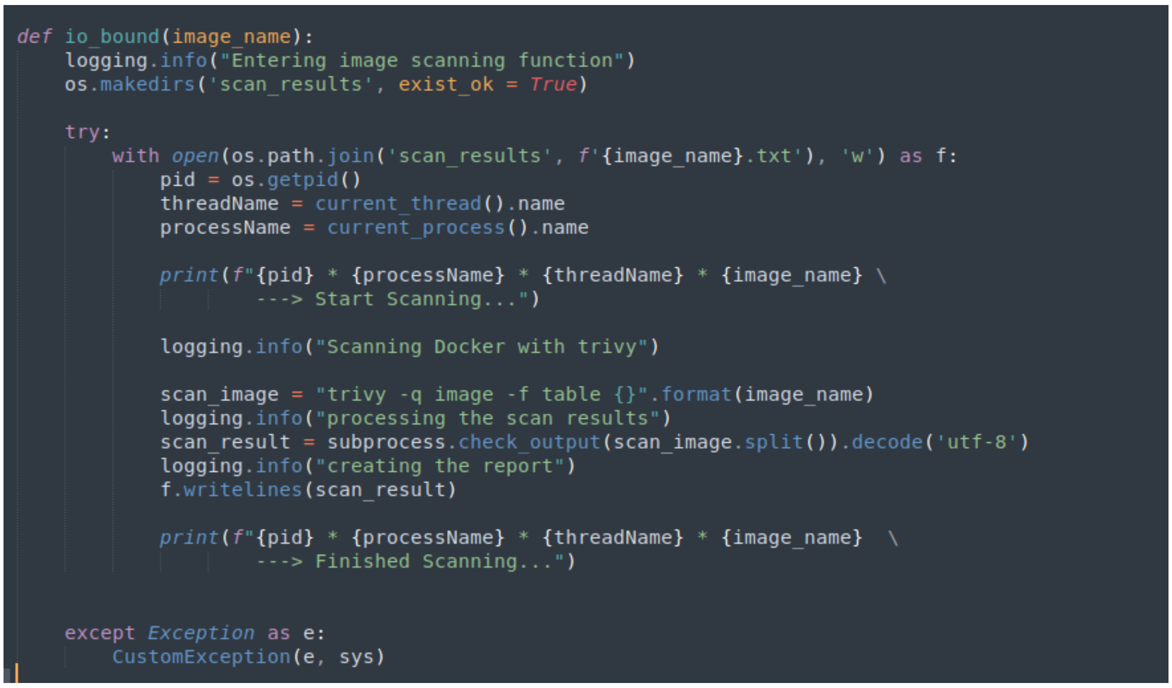


```
  GNU nano 7.2                                    docker_img_list.txt
image_1
image_2
```

Figure 3: Docker_img_list.txt

3.4 Tool Development:

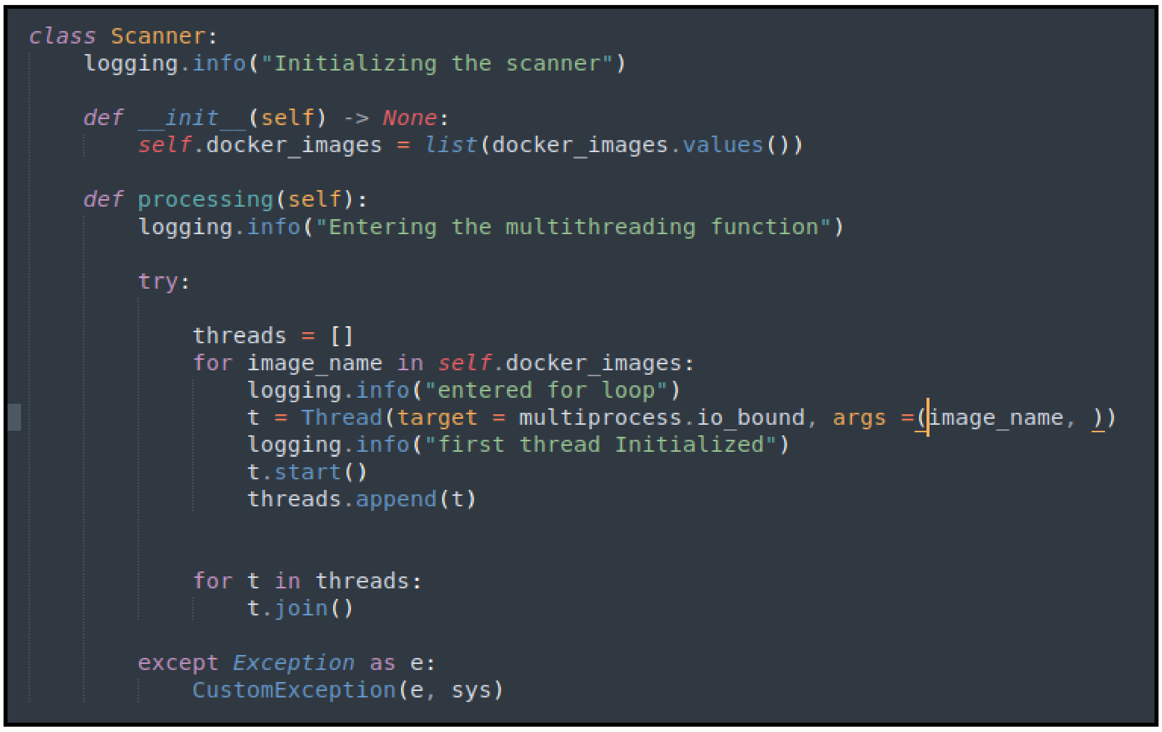
The multiprocess.py module had been created, which includes the scanning mechanism. Test images were used to thoroughly test the scanning mechanism. The scanning mechanism includes integration of tool i.e., Trivy. The scanned report is stored in the scan_results directory. To differentiate between the which container is scanning at the current time, use of processid, processName, threadName and image_name is utilized.



```python
def io_bound(image_name):
    logging.info("Entering image scanning function")
    os.makedirs('scan_results', exist_ok = True)

    try:
        with open(os.path.join('scan_results', f'{image_name}.txt'), 'w') as f:
            pid = os.getpid()
            threadName = current_thread().name
            processName = current_process().name

            print(f"{pid} * {processName} * {threadName} * {image_name} \
                    ---> Start Scanning...")

            logging.info("Scanning Docker with trivy")

            scan_image = "trivy -q image -f table {}".format(image_name)
            logging.info("processing the scan results")
            scan_result = subprocess.check_output(scan_image.split()).decode('utf-8')
            logging.info("creating the report")
            f.writelines(scan_result)

            print(f"{pid} * {processName} * {threadName} * {image_name}  \
                    ---> Finished Scanning...")


    except Exception as e:
        CustomException(e, sys)
```

Figure 4: Multiporcess.py

Then once the multiprocess.py is done, we created a logic for dynamically creating threads based on number of images passed. For achieving this, Threads module in python is used and the function from multiprocess module is called.



```python
class Scanner:
    logging.info("Initializing the scanner")

    def __init__(self) -> None:
        self.docker_images = list(docker_images.values())

    def processing(self):
        logging.info("Entering the multithreading function")

        try:

            threads = []
            for image_name in self.docker_images:
                logging.info("entered for loop")
                t = Thread(target = multiprocess.io_bound, args =(image_name, ))
                logging.info("first thread Initialized")
                t.start()
                threads.append(t)


            for t in threads:
                t.join()

        except Exception as e:
            CustomException(e, sys)
```

Figure 5: app.py

When it comes to thread safety, caution has been taken, to prevent race conditions. The use of multi-threading function, not only saves resource, but also save times.

3.5 Configure Automation for Jenkins:

A t2.meduim AWS instance was created with Ubuntu 20.04. On start , Shell scripts were used to install the necessary packages (Jenkins, Trivy, Docker).Jenkins was set-up such that it can be reached at <public_ip>:8080. EC2 instance on AWS offers the infrastructure needed to run the pipeline and host Jenkins. This guarantees that the pipeline has the tools it needs. EC2 instance's environment variables, networking, and security groups were configured.

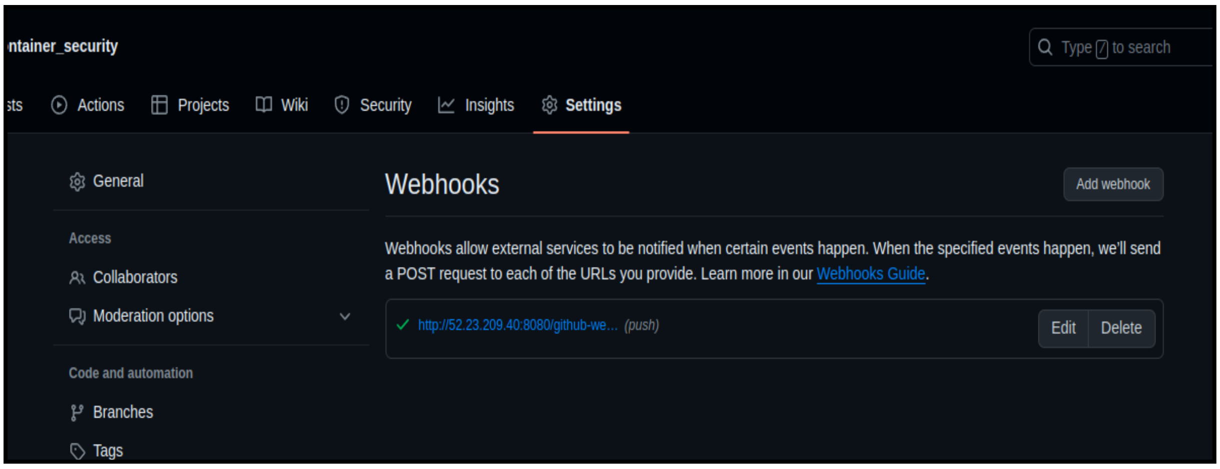
To manage pipeline, a new Jenkins task was created. In order to do this, project type was defined and connectted it to GitHub, version control system. The Groovy-written Jenkinsfile outlines the phases and actions in the pipeline. It is versioned on the GitHub repository, like the code.

Pipeline Stages:

- **SCM Finalization**: Take the materials and code out of the GitHub repository. In order to get the most recent code ready for the following steps, this stage retrieves it from the GitHub repository.
- **Image Build**: Use the Dockerfile to create Docker images. At this step, the Dockerfile is used to build the Docker images that are specified in your code.
- **Image Scan**: To perform vulnerability scanning, integrate Trivy. This is where Trivy integration takes place. Vulnerabilities are found in the Docker images created in the earlier phase.

3.6 GitHub Integration:

A GitHub repository was used to save code and artefacts. An automated triggering Jenkins pipeline using a GitHub webhook was set-up. The code and dockerfiles were kept in the GitHub repository. To start the Jenkins pipeline automatically anytime updates are pushed, a webhook in the GitHub repository was setup.



Webhooks

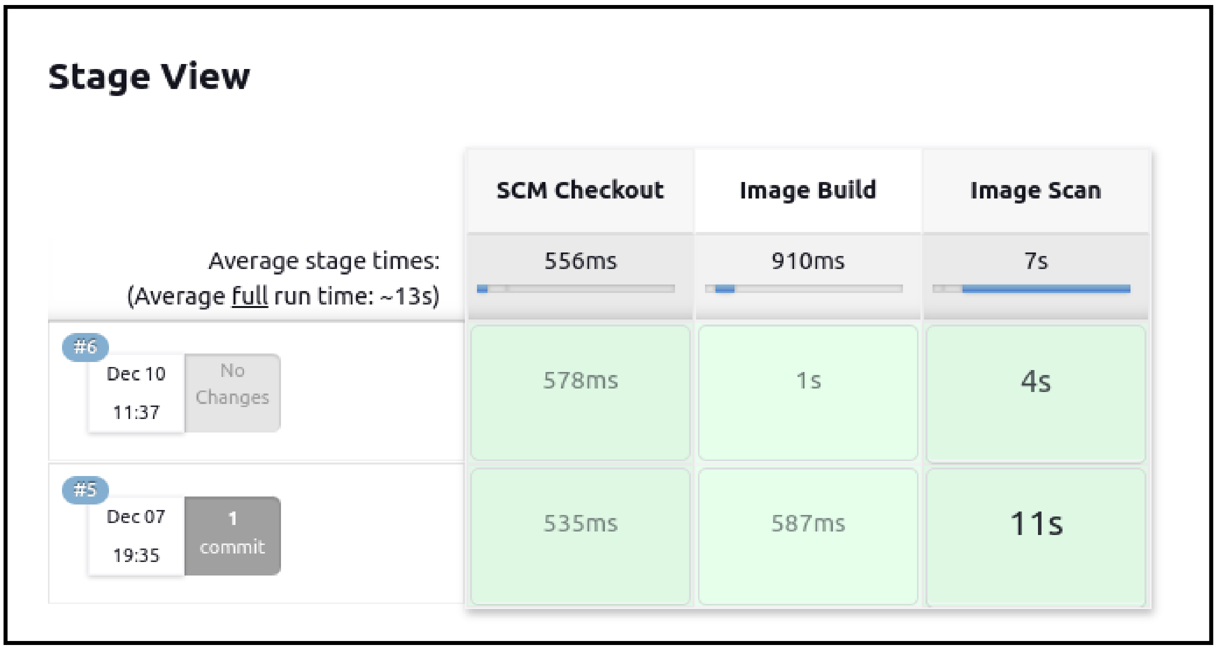
Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

✓ http://52.23.209.40:8080/github-we... (push)

Figure 6: GitHub Integration

3.7 Reviewing and Updating Pipeline:

We consistently checked Jenkins' pipeline setup. According to the project needs, the Jenkins file and pipeline steps were updated. It was ensured that Jenkins pipeline configuration is in line with the demands of the current project by reviewing it on a regular basis. As necessary, the Jenkins file was updated. The build and deployment processes accordingly modifies the file.



Stage View

	SCM Checkout	Image Build	Image Scan
Average stage times: (Average full run time: ~13s)	556ms	910ms	7s
#6 Dec 10 11:37 No Changes	578ms	1s	4s
#5 Dec 07 19:35 1 commit	535ms	587ms	11s

Figure 7: Jenkin's Pipeline

4 Design Specification

This section provides design specifications for the project. The scope of the specifications includes the development of essential modules such as logger.py and exception.py, the preparation of image lists from Docker_img_list.txt, the implementation of the scanning mechanism in multiprocess.py, the setup of Jenkins automation using an AWS EC2 instance, parallel scanning with multithreading, and continuous improvement practices. It makes use of Jenkins for continuous integration.

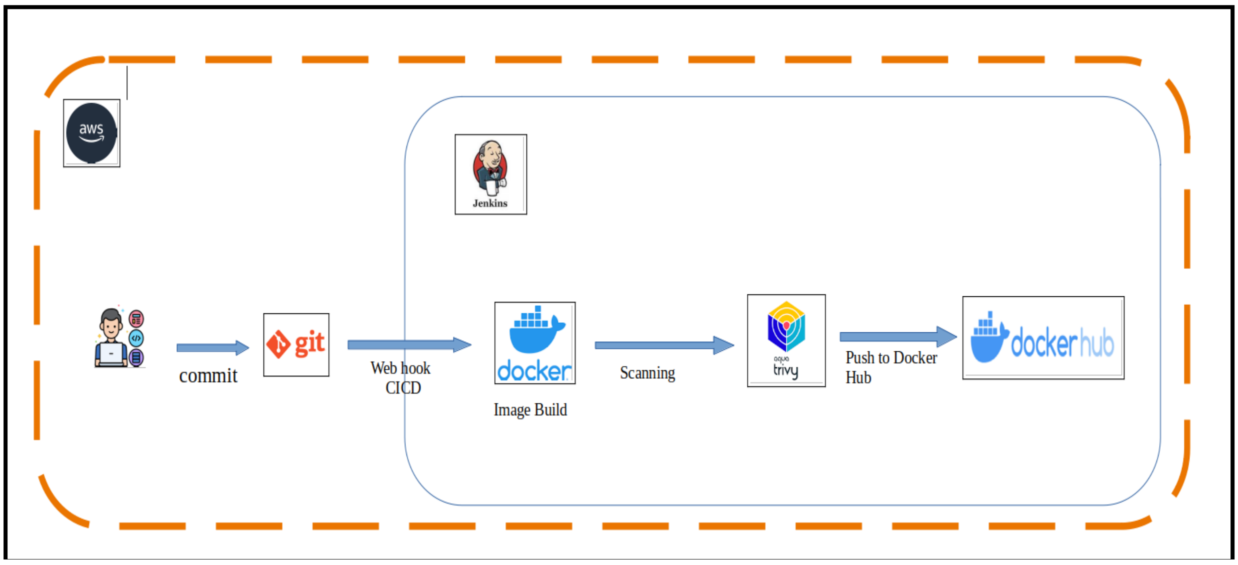


Figure 8: Architecture Diagram

4.1 Functional Requirements:

- Modules for Logging and Exception Handling:
  Specify custom exceptions for problems that arise during runtime. For efficient debugging, record events and failures. Put fallback plans in place in case of serious errors.

- Making an Image List:
  Conditions: Verify the format and existence of Docker_img_list.txt.
  List the images that need to be scanned after reading the file.

- Implementing the Scanning Mechanism:
  Develop scanning logic in multiprocess.py as per the requirements.
  Use a scanning tool (Trivy) to evaluate vulnerabilities.

- Multithreaded Parallel Scanning:
  Requirements: Use multiprocess.py to integrate scanning with multithreading.
  Make sure there is thread safety to avoid race situations.

- Configuring Jenkins Automation:
  Conditions: Get an Ubuntu 20.04 AMI running on an Amazon EC2 instance (t2.medium).
  To install Docker, Trivy, and Jenkins, write shell scripts.
  Set up environment variables, networking, and security groups.

- Ongoing Enhancement:
  Update tools and dependencies on a regular basis.
  Take testing and production deployment feedback into account.
  Keep an eye out for security updates and install them right away.

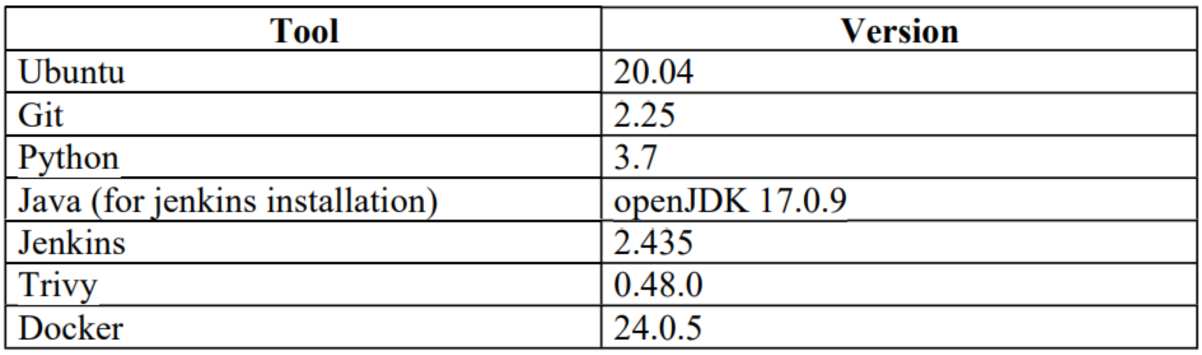
4.2 Non-Functional Requirements:

- Security
  To ensure secure access to Jenkins, use HTTPS.
  Avoid hardcoding credentials and instead store important information securely.
  Put authentication procedures and access limits in place.

- Usability:
  Create logging messages that are easy to read so that issues can be found quickly.

5 Implementation

Below is a synopsis of the main elements and procedures mentioned:



Tool	Version
Ubuntu	20.04
Git	2.25
Python	3.7
Java (for jenkins installation)	openJDK 17.0.9
Jenkins	2.435
Trivy	0.48.0
Docker	24.0.5

Table 2: Configuration

Sample Applications:

- The target_1 and target_2 directories include sample programmes that are probably meant to be used when building Docker images.
- These programmes are designed to be used for scanning.

Logging and Exception Handling:

- The modules exception.py and logger.py are used to handle exceptions and monitor the behavior of the application.

Docker Image Creation:

- The sample apps can be used to produce Docker images.
- Scripts, namely img_build.sh, are used to simplify the generation of images.

Docker Images Listing:

- A list of Docker images can be retrieved using the get_img.py module.

Using Multithreading to Scanning Images:

- A multithreading technique for simultaneously scanning two photos is implemented via the multiprocess.py package.

Image Scanning Procedure:

- The process of scanning images is made simpler by the scan_img.sh script.
- The app.py module, which combines multiple modules for multithreading, exception handling, and logging, is where the scanning technique is really implemented.

Using Multiple Processing to Scanning Docker Images:

- The app.py module combines the features of the previous modules to accomplish multiprocessing for Docker image scanning.

Using Multiple Processing to Scanning Docker Images:

- The app.py module combines the features of the previous modules to accomplish multiprocessing for Docker image scanning.

With a focus on streamlining these procedures using scripts and modular Python code, it offers an organized method for creating, inspecting, and maintaining Docker images.

An automated procedure for establishing a Jenkins continuous integration and deployment (CI/CD) pipeline for a project aimed at improving container security is described in the instructions that are included. The steps are summarized as follows:

EC2 Instance Configuration:

- Using the Ubuntu 20.04 AMI, create an EC2 t2.medium instance on Amazon.com.

Setting up Jenkins:

- Install Jenkins on the EC2 instance by running the jenkins.sh script.
- Java is needed by Jenkins, and this dependence is handled by the script.

Setting up Docker:

- Install Docker using the docker.sh script to provide the container runtime required for handling containerized images.

Setting up Trivy:

- Install Trivy, a tool for detecting vulnerabilities in containerized images, using the trivy.sh script.

Setting up the Jenkins Pipeline:

- Make a Jenkins pipeline job and use the GitHub source code.
- Configure the pipeline to use GitHub as the source code management platform.
- In the GitHub repository's settings, enable the "GitHub webhook" feature and set the Jenkins URL as the payload URL.

- Jenkins and GitHub are connected securely thanks to the webhook, and the connection is successful when shown by a green checkmark.

Automated Workflow for CI/CD:

- Jenkins runs automatically every time there is a code update in the GitHub repository.
- There are various stages in the pipeline:
- Gets the most recent updates from the GitHub repository during the Git Checkout stage.
- Build Stage: Using the supplied Dockerfile, builds the Docker image.
- Stage of Scanning: Trivy is used to search the containerized image for vulnerabilities.
- The text file with the scanning stage's results may be found in the "/var/lib/jenkins/workspace/enhance_container_security/" directory.

With this configuration, increased security protections are provided throughout the development lifecycle through an automated and continuous process of generating, testing, and scanning containerized images whenever changes are made to the codebase.

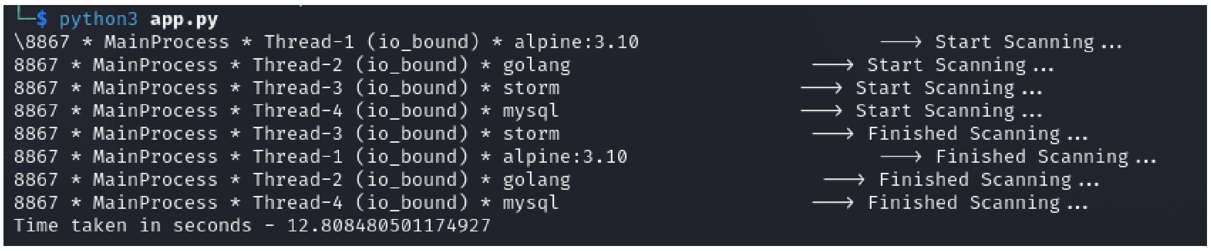
6 Evaluation

The Evaluation section analyses the methodology's outcomes to assess its efficacy in meeting defined goals around scan performance, resource optimization, and seamless integration. Key metrics examined include variation in scan duration, consistency in detection accuracy during concurrent scanning, resource utilization benefits, and qualitative measures of pipeline integration and automation. The analysis relies on experimental data gathered from the prototype implementation, including quantifiable timing data, scanning outputs, and infrastructure monitoring. By interpreting the result, conclusions are drawn regarding the methodology's strengths and limitations. The purpose of this section is to compare current practices with the solution for efficiency, security, and cost-effectiveness of containerised workloads.

6.1 Variation in Scan Duration

Evaluation: When compared to manual scanning, the methodology greatly shortens scan times. The efficiency achieved is demonstrated by the stated difference of 12.8 seconds for scanning 4 containers compared to 25–28 seconds for sequential scanning. This is a great advantage when there are a lot of containers in an organizational setting.

Strength: In situations where speed and efficiency are critical, the tool saves a significant amount of time and is a real asset in large-scale container setups.



```
└─$ python3 app.py
\8867 * MainProcess * Thread-1 (io_bound) * alpine:3.10            ──→ Start Scanning ...
8867 * MainProcess * Thread-2 (io_bound) * golang          ──→ Start Scanning ...
8867 * MainProcess * Thread-3 (io_bound) * storm          ──→ Start Scanning ...
8867 * MainProcess * Thread-4 (io_bound) * mysql          ──→ Start Scanning ...
8867 * MainProcess * Thread-3 (io_bound) * storm           ──→ Finished Scanning ...
8867 * MainProcess * Thread-1 (io_bound) * alpine:3.10            ──→ Finished Scanning ...
8867 * MainProcess * Thread-2 (io_bound) * golang            ──→ Finished Scanning ...
8867 * MainProcess * Thread-4 (io_bound) * mysql           ──→ Finished Scanning ...
Time taken in seconds - 12.808480501174927
```

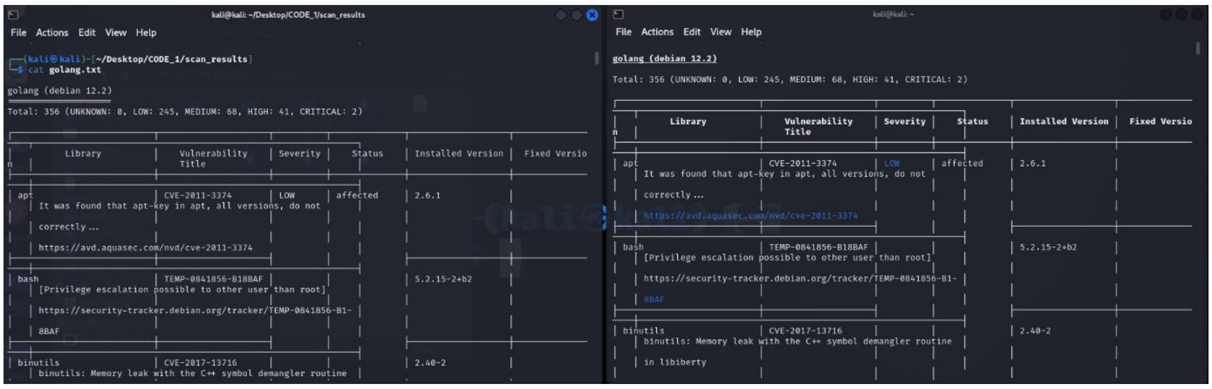
Figure 9: Time taken for scanning 4 containers

6.2 Variation in the Total Number of Vulnerabilities Found

Evaluation: The methodology shows consistency in discovering the same number of vulnerabilities across numerous scans run, which solves a prevalent concern in concurrent scanning technologies. This suggests that the accuracy of vulnerability detection is not jeopardized by the parallel scanning approach.

Strength: The tool's trustworthiness is increased by its capacity to consistently identify vulnerabilities, which guarantees that organizations can depend on reliable results even when using concurrent scanning.

The result on left-hand side is generated by the code and the result on the right-hand side is generated by manually scanning the container using trivy via terminal.


kali@kali: ~/Desktop/CODE_1/scan_results
File Actions Edit View Help
kali@kali: ~
File Actions Edit View Help

```
┌──(kali㉿kali)-[~/Desktop/CODE_1/scan_results]
└─$ cat golang.txt

golang (debian 12.2)

Total: 356 (UNKNOWN: 0, LOW: 245, MEDIUM: 68, HIGH: 41, CRITICAL: 2)

│   Library        │ Vulnerability │ Severity │ Status   │ Installed Version │ Fixed Versio
n │                │ Title         │          │          │                   │
│ apt              │ CVE-2011-3374 │ LOW      │ affected │ 2.6.1             │
  │ It was found that apt-key in apt, all versions, do not
  │ correctly ...
  │ https://avd.aquasec.com/nvd/cve-2011-3374
│ bash             │ TEMP-0841856-B18BAF │    │          │ 5.2.15-2+b2       │
  │ [Privilege escalation possible to other user than root]
  │ https://security-tracker.debian.org/tracker/TEMP-0841856-B1-
  │ 8BAF
│ binutils         │ CVE-2017-13716 │         │          │ 2.40-2            │
  │ binutils: Memory leak with the C++ symbol demangler routine
```

```
golang (debian 12.2)

Total: 356 (UNKNOWN: 0, LOW: 245, MEDIUM: 68, HIGH: 41, CRITICAL: 2)

│   Library        │ Vulnerability │ Severity │ Status   │ Installed Version │ Fixed Versio
n │                │ Title         │          │          │                   │
│ apt              │ CVE-2011-3374 │ LOW      │ affected │ 2.6.1             │
  │ It was found that apt-key in apt, all versions, do not
  │ correctly ...
  │ https://avd.aquasec.com/nvd/cve-2011-3374
│ bash             │ TEMP-0841856-B18BAF │    │          │ 5.2.15-2+b2       │
  │ [Privilege escalation possible to other user than root]
  │ https://security-tracker.debian.org/tracker/TEMP-0841856-B1-
  │ 8BAF
│ binutils         │ CVE-2017-13716 │         │          │ 2.40-2            │
  │ binutils: Memory leak with the C++ symbol demangler routine
  │ in libiberty
```

Figure 10: Scan Result

6.3 Pipeline Automation for CI/CD

Evaluation: One notable strength is the integration of pipeline automation for CI/CD. The solution automatically initiates scans with every repository modification and blends in smoothly with the development workflow. This illustrates how security checks may be smoothly incorporated into deployment and continuous integration procedures in the real world.

Strength: Automation promotes a proactive security posture by increasing efficiency and guaranteeing that security checks are an essential component of the development lifecycle.

6.4 Conserving Resources

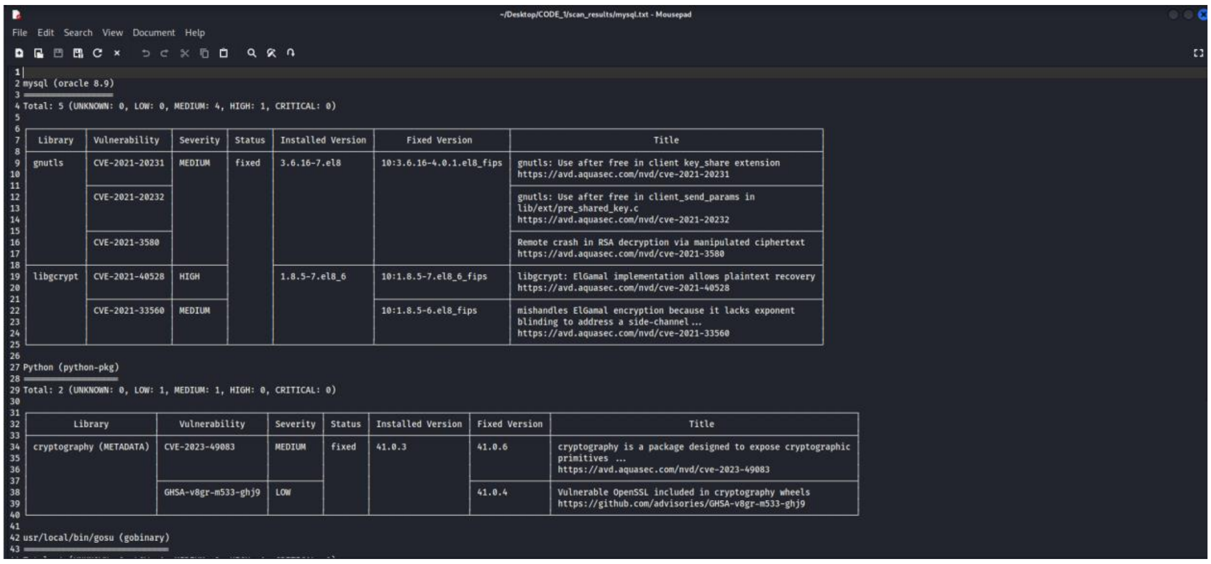
Evaluation: The tool's design, which uses threads for parallel scanning and operates on a single machine, shows to be resource-efficient. Without requiring the installation of tools on each container or the creation of multiple tool instances, resource utilization is optimized by the dynamic generation of threads based on the number of containers.

Strength: Scalability is ensured by the resource-efficient design, which allows the tool to adjust to changing workloads and available processing power.

6.5 Report Formatted Based on Criticality of Vulnerability

Evaluation: Based on CVSS ratings, multiple severity levels are represented in the report created by Trivy. This organized output improves the tool's usability and practical insights by helping to priorities and address vulnerabilities based on their criticality.

Strength: A clear and standardized approach for ranking and addressing vulnerabilities is provided by the integration of CVSS scores with the prepared report, which helps to facilitate successful remediation efforts.



~/Desktop/CODE_1/scan_results/mysql.txt - Mousepad

```

mysql (oracle 8.9)
==================
Total: 5 (UNKNOWN: 0, LOW: 0, MEDIUM: 4, HIGH: 1, CRITICAL: 0)


┌───────────┬────────────────┬──────────┬────────┬───────────────────┬───────────────────────────┬────────────────────────────────────────────────────────────┐
│  Library  │ Vulnerability  │ Severity │ Status │ Installed Version │       Fixed Version       │                           Title                            │
├───────────┼────────────────┼──────────┼────────┼───────────────────┼───────────────────────────┼────────────────────────────────────────────────────────────┤
│ gnutls    │ CVE-2021-20231 │ MEDIUM   │ fixed  │ 3.6.16-7.el8      │ 10:3.6.16-4.0.1.el8_fips  │ gnutls: Use after free in client key_share extension       │
│           │                │          │        │                   │                           │ https://avd.aquasec.com/nvd/cve-2021-20231                 │
│           ├────────────────┤          │        │                   │                           ├────────────────────────────────────────────────────────────┤
│           │ CVE-2021-20232 │          │        │                   │                           │ gnutls: Use after free in client_send_params in            │
│           │                │          │        │                   │                           │ lib/ext/pre_shared_key.c                                   │
│           │                │          │        │                   │                           │ https://avd.aquasec.com/nvd/cve-2021-20232                 │
│           ├────────────────┤          │        │                   │                           ├────────────────────────────────────────────────────────────┤
│           │ CVE-2021-3580  │          │        │                   │                           │ Remote crash in RSA decryption via manipulated ciphertext  │
│           │                │          │        │                   │                           │ https://avd.aquasec.com/nvd/cve-2021-3580                  │
├───────────┼────────────────┼──────────┤        ├───────────────────┼───────────────────────────┼────────────────────────────────────────────────────────────┤
│ libgcrypt │ CVE-2021-40528 │ HIGH     │        │ 1.8.5-7.el8_6     │ 10:1.8.5-7.el8_6_fips     │ libgcrypt: ElGamal implementation allows plaintext recovery │
│           │                │          │        │                   │                           │ https://avd.aquasec.com/nvd/cve-2021-40528                 │
│           ├────────────────┼──────────┤        │                   ├───────────────────────────┼────────────────────────────────────────────────────────────┤
│           │ CVE-2021-33560 │ MEDIUM   │        │                   │ 10:1.8.5-6.el8_fips       │ mishandles ElGamal encryption because it lacks exponent    │
│           │                │          │        │                   │                           │ blinding to address a side-channel ...                     │
│           │                │          │        │                   │                           │ https://avd.aquasec.com/nvd/cve-2021-33560                 │
└───────────┴────────────────┴──────────┴────────┴───────────────────┴───────────────────────────┴────────────────────────────────────────────────────────────┘


Python (python-pkg)
===================
Total: 2 (UNKNOWN: 0, LOW: 1, MEDIUM: 1, HIGH: 0, CRITICAL: 0)


┌──────────────────────────┬─────────────────────┬──────────┬────────┬───────────────────┬───────────────┬─────────────────────────────────────────────────────────────┐
│         Library          │    Vulnerability    │ Severity │ Status │ Installed Version │ Fixed Version │                            Title                            │
├──────────────────────────┼─────────────────────┼──────────┼────────┼───────────────────┼───────────────┼─────────────────────────────────────────────────────────────┤
│ cryptography (METADATA)  │ CVE-2023-49083      │ MEDIUM   │ fixed  │ 41.0.3            │ 41.0.6        │ cryptography is a package designed to expose cryptographic  │
│                          │                     │          │        │                   │               │ primitives ...                                              │
│                          │                     │          │        │                   │               │ https://avd.aquasec.com/nvd/cve-2023-49083                  │
│                          ├─────────────────────┼──────────┤        │                   ├───────────────┼─────────────────────────────────────────────────────────────┤
│                          │ GHSA-v8gr-m533-ghj9 │ LOW      │        │                   │ 41.0.4        │ Vulnerable OpenSSL included in cryptography wheels          │
│                          │                     │          │        │                   │               │ https://github.com/advisories/GHSA-v8gr-m533-ghj9           │
└──────────────────────────┴─────────────────────┴──────────┴────────┴───────────────────┴───────────────┴─────────────────────────────────────────────────────────────┘


usr/local/bin/gosu (gobinary)
=============================
```

Figure 11: Report File

6.6 Discussion

The results demonstrate that the developed methodology for automated parallel scanning of Docker images provides significant improvements in efficiency and resource utilization compared to sequential scanning approaches. Specifically, the scan time for 4 containers was reduced from 25-28 seconds using sequential approach down to just 12.8 seconds with the concurrent approach. This represents over 40%-time savings, which would enable much faster security scanning in contexts with large numbers of containers. The consistency in the number of vulnerabilities detected across repeated scans also shows that the accuracy of detection is not compromised by introducing parallelism, ensuring reliability.

Additionally, the single-machine design using dynamic thread allocation also allows efficient scaling without requiring multiple tool instances or installation on every container.

Together, these benefits clearly demonstrate more optimized resource usage, improved speed, and stronger security posture. However, some limitations exist in the current implementation. The demonstration the approach was only on a small number of containers. Testing on larger, more complex repositories would better validate scalability. The prototype pipeline also relies on a specific set of technologies like Jenkins, Docker, and Trivy. Integration with additional orchestrators like Kubernetes or other scanning tools may improve adaptability across diverse infrastructures.

Nonetheless, within the defined scope, the methodology delivers on the stated goals of faster scanning, lower resource footprint, and tighter integration with CI/CD automation. These capabilities directly address pressing needs for container workload security, preventing organizations from having to choose between velocity and protection. With further enhancement and expanded testing, the techniques show promise for wide adoption securing cloud-native development.

7 Conclusion and Future Work

This research presented an automated methodology for efficient parallel scanning of Docker container images to enhance security. The implementation demonstrates significantly faster scan times compared to sequential approaches, reducing the duration by over 40% for a small set of sample containers. The accuracy of vulnerability detection is also consistent across repeated parallelized scans.

Additionally, the integration of the scanning pipeline with CI/CD automation using Jenkins enables security checks to be intrinsically embedded within development workflows. Scans automatically trigger with code changes to proactively detect vulnerabilities early in the lifecycle. The single-machine architecture also optimizes resource utilization by dynamically allocating threads based on workload instead of needing isolated tools.

Together, these capabilities improve the speed, cost-effectiveness and security posture of organizations relying on containerized workloads. Developers gain the ability to move fast without compromising protection.

While showing promise, further enrichment of the implementation and testing on larger, real-world repositories would be beneficial future work. Expanding compatibility with additional orchestrators like Kubernetes and integrating dynamic scanning tools could also make the approach viable across more diverse infrastructures. Exploring other methods like machine learning to prioritize scan targets could further optimize the pipeline.

Overall, this research makes valuable progress in addressing the growing need for efficient security solutions tailored to container environments. By combining automation, parallelization and tight integration with development pipelines, key advantages are achieved in performance, scalability and organizational fit. These techniques lay groundwork for more innovations that secure container adoption across industries. With additional work, the methodology has potential for widespread impact securing cloud-native applications.

References

Alqarni, A. (2023). *Enhancing Cloud Security and Privacy With Zero-Knowledge Encryption and Vulnerability Assessment in Kubernetes Deployments - ProQuest*. [online] www.proquest.com. Available at: https://www.proquest.com/openview/5f3859b286536da5318f915d68e87666/1?pq-origsite=gscholar&cbl=18750&diss=y [Accessed 13 Dec. 2023].

Amith Raj MP, Kumar, A., Pai, S.J. and Gopal, A. (2016). *Enhancing security of Docker using Linux hardening techniques*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICATCCT.2016.7911971.

Andersson, M. and Hysing Berg, R. (2022). *http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1679447*. [online] DIVA. Available at: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1679447&dswid=7896 [Accessed 13 Dec. 2023].

Berkovich, S., Kam, J. and Wurster, G. (2020). *{UBCIS}: Ultimate Benchmark for Container Image Scanning*. [online] www.usenix.org. Available at: https://www.usenix.org/conference/cset20/presentation/berkovich [Accessed 13 Dec. 2023].

Bui, T. (2015). Analysis of Docker Security. *arXiv:1501.02967 [cs]*. [online] Available at: https://arxiv.org/abs/1501.02967.

Casalicchio, E. (2016). Autonomic Orchestration of Containers: Problem Definition and Research Challenges. doi:https://doi.org/10.1145/12345.67890.

Chan, S. (2018). *Prototype Open-Source Software Stack for the Reduction of False Positives and Negatives in the Detection of Cyber Indicators of Compromise and Attack: Hybridized Log Analysis Correlation Engine and Container-Orchestration System Supplemented by Ensemble Method Voting Algorithms for Enhanced Event Correlation*. [online] Ssrn.com. Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3789766 [Accessed 13 Dec. 2023].

Čilić, I., Krivić, P., Podnar Žarko, I. and Kušek, M. (2023). Performance Evaluation of Container Orchestration Tools in Edge Computing Environments. *Sensors*, [online] 23(8), p.4008. doi:https://doi.org/10.3390/s23084008.

Hoque, S., Brito, M.S. de, Willner, A., Keil, O. and Magedanz, T. (2017). Towards Container Orchestration in Fog Computing Infrastructures. *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*. [online] doi:https://doi.org/10.1109/compsac.2017.248.

Javed, O. and Toor, S. (2021). *Understanding the Quality of Container Security Vulnerability Detection Tools*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2101.03844.

Lumio, N. (2018). *DSpace*. [online] aaltodoc.aalto.fi. Available at: https://aaltodoc.aalto.fi/items/b977a10c-e8b0-40aa-99ee-f4e56e4d40b4 [Accessed 13 Dec. 2023].

Martin, A., Raponi, S., Combe, T. and Di Pietro, R. (2018). Docker ecosystem – Vulnerability Analysis. *Computer Communications*, 122, pp.30–43. doi:https://doi.org/10.1016/j.comcom.2018.03.011.

Mavridis, I. and Karatza, H. (2021). Orchestrated sandboxed containers, unikernels, and virtual machines for isolation-enhanced multitenant workloads and serverless computing in cloud. *Concurrency and Computation: Practice and Experience*, 35.11(p.e6365). doi:https://doi.org/10.1002/cpe.6365.

Moreau, D., Wiebels, K. and Boettiger, C. (2023). Containers for computational reproducibility. *Nature Reviews Methods Primers*, 3(1). doi:https://doi.org/10.1038/s43586-023-00236-9.

Morkevicius, N., Venčkauskas, A., Šatkauskas, N. and Toldinas, J. (2021). Method for Dynamic Service Orchestration in Fog Computing. *Electronics*, 10(15), p.1796. doi:https://doi.org/10.3390/electronics10151796.

Pahl, C., Brogi, A., Soldani, J. and Jamshidi, P. (2019). Cloud Container Technologies: A State-of-the-Art Review. *IEEE Transactions on Cloud Computing*, [online] 7(3), pp.677–692. doi:https://doi.org/10.1109/tcc.2017.2702586.

Perera, H.P.D.S., Reza, B., De Silva, H.S.T., Karunarathne, A.D.H.U., Ganegoda, B. and Senarathne, A. (2022). Docker Container Security Orchestration and Posture Management Tool. *2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. doi:https://doi.org/10.1109/icccnt54827.2022.9984287.

R. Saminathan V. Mahavaishnavi (2023). Container Security Intelligence: Leveraging Machine Learning for Anomaly Detection in Containerized Applications. *Tuijin Jishu/Journal of Propulsion Technology*, 44(3), pp.3717–3730. doi:https://doi.org/10.52783/tjjpt.v44.i3.2091.

Raj, P. and Raman, A. (2018). Automated Multi-cloud Operations and Container Orchestration. *Software-Defined Cloud Centers*, pp.185–218. doi:https://doi.org/10.1007/978-3-319-78637-7_9.

Ranaweera, P., Jurcut, A.D. and Liyanage, M. (2021). Survey on Multi-Access Edge Computing Security and Privacy. *IEEE Communications Surveys & Tutorials*, 23(2), pp.1078–1124. doi:https://doi.org/10.1109/comst.2021.3062546.

Şenel, B. (2023). *Container Orchestration for the Edge Cloud*. [online] theses.hal.science. Available at: https://theses.hal.science/tel-04197683/ [Accessed 13 Dec. 2023].

Sharma, P., Chaufournier, L., Shenoy, P. and Tay, Y.C. (2016). Containers and Virtual Machines at Scale. *Proceedings of the 17th International Middleware Conference*. doi:https://doi.org/10.1145/2988336.2988337.

Sultan, S., Ahmad, I. and Dimitiou, T. (2019). *Container Security: Issues, Challenges, and the Road Ahead | IEEE Journals & Magazine | IEEE Xplore*. [online] ieeexplore.ieee.org. Available at: https://ieeexplore.ieee.org/abstract/document/8693491 [Accessed 13 Dec. 2023].

Tiwari, H. (2023). *Enhancing Container Security Through Automated Vulnerability Scanning and Remediation with Trivy*. [online] Insights2Techinfo. Available at: https://insights2techinfo.com/enhancing-container-security-through-automated-vulnerability-scanning-and-remediation-with-trivy/ [Accessed 13 Dec. 2023].

V s, D.P., Chakkaravarthy Sethuraman, S. and Khan, M.K. (2023). Container security: Precaution levels, mitigation strategies, and research perspectives. *Computers & Security*, [online] 135, p.103490. doi:https://doi.org/10.1016/j.cose.2023.103490.

Vaño, R., Lacalle, I., Piotr Sowiński, Raúl S-Julián and Palau, C.E. (2023). Cloud-Native Workload Orchestration at the Edge: A Deployment Review and Future Directions. *Sensors*, 23(4), pp.2215–2215. doi:https://doi.org/10.3390/s23042215.

Xu, X., Xu, A., Jiang, Y., Jain, V., Singh, B., Khenwar, M. and Sharma, M. (2021). Static Vulnerability Analysis of Docker Images You may also like Research on Security Issues of Docker and Container Monitoring System in Edge Computing System Static Vulnerability Analysis of Docker Images. *IOP Conference Series: Materials Science and Engineering*. doi:https://doi.org/10.1088/1757-899X/1131/1/012018.

Zhu, H. and Gehrmann, C. (2021). Lic-Sec: An enhanced AppArmor Docker security profile generator. *Journal of Information Security and Applications*, 61, p.102924. doi:https://doi.org/10.1016/j.jisa.2021.102924.

Zhu, H., Gehrmann, C. and Roth, P. (2023). Access Security Policy Generation for Containers as a Cloud Service. *SN Computer Science*, 4(6). doi:https://doi.org/10.1007/s42979-023-02186-1.