National College of Ireland

Detecting Container vulnerabilities leveraging the CICD pipeline

MSc Research Project
Cybersecurity

Preeti Bhardwaj
Student ID: x21139351

School of Computing
National College of Ireland

Supervisor: Arghir Nicolae Moldovan

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Preeti Bhardwaj
Student ID:	x21139351
Programme:	Cybersecurity
Year:	2022
Module:	MSc Research Project
Supervisor:	Arghir Nicolae Moldovan
Submission Due Date:	15/12/2022
Project Title:	Detecting Container vulnerabilities leveraging the CICD pipeline
Word Count:	7126
Page Count:	23

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:	Preeti Bhardwaj
Date:	29th January 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):	

Detecting Container vulnerabilities leveraging the CICD pipeline

Preeti Bhardwaj
x21139351

Abstract

Docker images are lightweight executable software that contains everything that is needed to make an application run smoothly. Docker images are a popular choice over virtual machines. Docker images have multiple vulnerabilities in them such as Denial-of-services(DoS), Man-in-Middle, etc, which makes it essential to test and secure these images effectively. There are several tools to test the docker images Perhaps the limitation comes down to the methodology which in most cases might not be automated and can be tedious to carry out. Scanning the docker images for vulnerabilities is essential as any non-detected vulnerability leaked from the image can corrupt the host system, can lead to the insertion of malicious code in the image, or can even spoof MAC and IP addresses. We have implemented a CI CD pipeline with the capability to scan the docker images for CVEs in an automated test bed for vulnerabilities before deploying them. The setup consists of a pipeline with the different stages from build to test along with the tools such as Trivy and Clair to scan the vulnerabilities on the docker images with known vulnerabilities and CVEs.

1 Introduction

Docker containerisation is the method of packaging the applications requirements into a single object file consisting of the application's libraries, binaries and code. Which makes containers a suitable choice for deployment by a lot of companies and organisations. Unlike traditional computing, virtualization does not depend on the physical hardware, firmware or the host which resulted in prolonged and underutilised cycles. Manu et al. (2016) introduced the concept of virtualization in the 1970's to overshadow the monolithic traditional computing practices, later to become the core of cloud computing. Virtualization requires an Operating system supported by virtual hardware resources. On the other hand containerization uses different technology in comparison to the classic virtualization making a better usage of RAM and CPU processing. Containers consume only the operating system with relevant libraries to host a program. The deployment process is relatively faster and lightweight on docker containers. Applications that require scalability and high performance can be efficiently deployed through container dockers. For the research Docker images are being used as the containerisation technology for the research implementation. Giant organisations such as Google, Amazon are leveraging docker services especially in their PaaS service model. Open-source docker behaves as a double-edge sword if security is considered. Docker images are vulnerable and are susceptible to attacks which need immediate attention. A docker image infected with malware/viruses

can act as a backdoor for the malicious activities by the attackers to exploit the image. Reportedly 60% of organisations suffered some kind of attack on their docker Profile (2016) in which using insecure docker images has been one of the biggest reasons.In May 2019 a shocking discovery was made which claimed that some Alpine docker images were shipped with no password henceforth as a backdoor for attackers.Seals and Seals (2015) Scanning docker images for vulnerabilities can be an effective step toward hardening the security of docker images. This research has worked towards the hardening docker images security by implementing an automated setup. In this research we implemented an automated CI CD pipeline which pulls the docker images from the repository and then the pipelines run with different stages which are: build, test and deploy. During the test stage we prepared a test bed by incorporating docker scanning tools: Clair and Trivy, both the tools scanned the vulnerable docker images for known Common vulnerabilities and Exposures(CVEs). Once after scanning the corrupt docker image we evaluated our result based on the known vulnerabilities. We evaluated the results of Clair and Trivy on 2 images to check to analyse how the two tools scan on the images and how different outputs are produced by both the tools on the basis of the number of vulnerabilities found and also the severity.

Contributions by the research paper:

- This paper contributed in successfully scanning two different Vulnerable docker images by Clair and Trivy.
- The pipeline used to achieve the scanning results is entirely automated which is an efficient contribution for an extensive implementation and process like this.
- We found Clair being used to scan for vulnerabilities on docker however Trivy is however is uncommon and we could not find any extensive research or analysis performed on Trivy. Our research is imperative to analyse the results of Trivy tool on docker in an automated pipeline to harden docker image security.
- Another big contribution is statistical evaluation. While reviewing the literature work we realised the lack of evaluation on the implementation results which we tried to achieve in our proposal work.

The later part of this research has been divided and organised into sections. Section [2] reflects on the research objective, related work in the field is discussed in section [3], section[4] focuses on the research methodology followed to carry out the research. The design and implementation has been specified in section [5] and [6]. The analysis of the outcome and evaluation cover section [7]. The research paper has been concluded in section [8] along with the future expectation.

Research question: How Container vulnerabilities be detected leveraging the CICD pipeline?

2 Objectives

The key objective of the research and the technique being used are as follows:

- To implement a test bed for scanning the docker images.

- To automate the process from pulling the docker image and scan them for known vulnerabilities of critical, high, medium and low severity.
- Automating the entire procedure is one of the other essential goals of the research.
- To evaluate the results and outcome of Clair and Trivy on multiple images.

3 Related Work

Cloud computing is a shared responsibility, the security in the cloud is a shared responsibility too of the cloud provider and user. Virtualization is the key enabler of cloud computing which supports services such as resource sharing, storage, network and data among different users and stakeholders. Virtualization provides allocation of on-demand cloud infrastructure services. Still traditional virtualization techniques have many shortcomings and limitations such as cost, cost of assembling, scalability, all these limitations were tried to be solved with the containerization. A single command can be used to scale up or down or deploy the containers within seconds. Docker ensures consistency over several different platforms and environments. When it comes to containers, docker is a popular choice for docker services among organisations. The security concerns and vulnerabilities pose a great threat to the underlying system or the environment they are deployed over. There are several factors responsible for security issues in docker images, one being the corrupt image which is not scanned for vulnerabilities prior to deployment. In this research we have followed an automated approach to scan the vulnerable docker images in continuous integration and continuous deployment pipeline.

In this section of the paper, several research literature has been deeply analysed for the purpose of literature review to understand the already existing work in this field of Devsec-ops and how our research is niche in terms of the methodology, research approach and tools etc. The one question that can be put up over here is that it depends on the source from where the docker image in question is being extracted, perhaps as per the studies University et al. (2017a) where almost 356,218 images were examined from both community and official repository. Henceforth, the challenge of selecting an image for scanning becomes a critical aspect of the research. University et al. (2017a) focused their study on docker images and had some important findings, the study claimed to have found almost 180 vulnerabilities in the images. Beside this, they also found out how vulnerabilities from parent images can transfer to child images. The study revealed a lot of security concerns in docker images which can be considered as one of the key evidence for the need to develop, find and implement some mechanism to secure docker images by scanning them in the development cycle.

Another study of Computer Forensics et al. (2017) focuses on the security issues on the fact that docker containers possess a kernel shared architecture which makes it easy for external malicious security issues to penetrate into the containers. Yang et al. (2021) also deep dives into the security challenges of containers at different layers in cloud environments mainly at container, kernels and orchestration layer. The solution and protection discussed in the study revolves around sandboxing, using namespaces and limiting resource usage however it lacks any solution or any mechanism to secure docker container images.There have been several other studies regarding the docker security, one such security issue has been addressed in Combe et al. (2016), an Orchestrator solution was proposed to solve some of the security however the study lacks any evidence of the

Docker Vulnerability Scanning Techniques					
Related Work	Docker	Detection Tech	Evaluation	Clair	Trivy
University et al. (2017a)	✓	✓			
of Computer Forensics et al. (2017)	✓	✓	✓		
Combe et al. (2016)	✓				
Wenhao and Zheng (2020)	✓				
Pathirathna et al. (2017)		✓	✓		
Pinnamaneni et al. (2022)	✓	✓	✓		
Yang et al. (2021)	✓	✓	✓		
Sengupta et al. (2021)	✓	✓			
Reeves et al. (2021)	✓	✓	✓		
Singh et al. (2019)	✓	✓	✓		
Mahajan and Mane (2022)	✓	✓			
University et al. (2017b)		✓	✓	✓	
Abhishek and Rajeswara Rao (2021)	✓	✓	✓		
Duarte and Antunes (2018)	✓	✓	✓		
Shameem Ahamed et al. (2021)	✓	✓			
Brady et al. (2020)	✓	✓	✓	✓	
Kwon and Lee (2020a)	✓	✓			
Tunde-Onadele et al. (2019)	✓	✓	✓	✓	

Table 1: Classification of Researches based on scanning techniques

testing of the solution. Wenhao and Zheng (2020) points out the vulnerabilities in docker on several aspects and how the vulnerabilities different on the basis of communication among the host and images, image transmission and system isolation. However the vulnerability research lacks audit and mechanism to exploit or scan the present security vulnerabilities.

The work carried out in Pathirathna et al. (2017) proposes a testing framework for the developers to scan their docker images having little to no knowledge about the entire testing process however the framework has been set leveraging Kubernetes mainly and lacks the deep analysis over docker, along with detention techniques in the work by Mahajan and Mane (2022) on kubernetes. Pinnamaneni et al. (2022) uses machine learning techniques to implement a mechanism to scan docker images however the implementation was not successful to find some high priority vulnerabilities in the image such as DoS which were evident in the docker image, in our research we have tried to implement an automated testbed which should be able to scan at least the evident high priority CVEs. Sengupta et al. (2021) Study shows another approach to harden the security of docker containers where an application is built to incorporate the mandatory security in the docker containers, however the approach to harden the security of the containers is quite different then the approach and technique we have taken into consideration , our research is focused on an active testing environment where the existing vulnerabilities on a docker container images can be detected and later can be mitigated for better security.

As discussed earlier there could be multiple vectors for container vulnerabilities, the study in Reeves et al. (2021) has covered container kernel bugs, run time bugs and mis configuration. The research is conducted on 59 run time vulnerabilities CVEs, the study lacks the image vulnerabilities. Our study focused on image vulnerabilities CVEs. Our research in question is implemented using the Gitlab CICD pipeline, study in Singh et al. (2019) has covered the integration of different tools in Gitlab CICD. Code integration to automate the process of development, building, testing and deployment, the comparison of Gitlab CICD and jenkins for better usage has been made, in our project we are using Gitlab CICD with integrated testing tools in it to automate the entire process of vulnerability scanning.

Static analysis is crucial in detecting vulnerabilities and malicious viruses at an early stage of the development process. In our research we have built a test bed in CICD pipeline with scanning tools integrated in the automated pipeline to harden the security scanning of the vulnerable docker images. In paper University et al. (2017b) a frame has been provided named as DIVA, the tool to scan statically used in the framework is Clair, Tunde-Onadele et al. (2019) used Clair tool only, however in our setup we have used Clair and Trivy to implement the best possible test bed for scanning vulnerable images. The DIVA tool only scanned for docker community repositories and their corresponding images, in our research we scanned for images from unofficial sources as well to understand the state of vulnerabilities on the images. Another difference is the approach and mechanism used to carry out the entire research process, automating the process is one of the major striking distinguishing factors. Similar research work and study has been shown in Abhishek and Rajeswara Rao (2021) where tools like SonarQube and VirusTotal have been used unlike our setup. Compared to our work the implementation and evaluation lacks an architectural security framework, a better approach could be to incorporate security at each layer.

Another similar work in Duarte and Antunes (2018) is depicted where not just the scanning using SCA tools were performed, some patch work work was also analysed. The

research revealed how the SCA tools were not effective enough, in our research we have tried to use a combination of tools which could possibly give some meaningful results. The tools used in our work can easily be manipulated in any language unlike in Duarte and Antunes (2018) which was limited to one programming language. Shameem Ahamed et al. (2021) categorised the images based on operating system, component and language. Later security issues were identified on the images which can help in security audits; perhaps there was no solution for the vulnerability. There is no mechanism provided in the study for the audit. Our research is backed by Brady et al. (2020) in which auditing tools such as Anchore has been used for testing docker images in Brady et al. (2020) on images from docker official registry,our research will be exploring other tools from unofficial repository. A model named DIVDS is proposed by Kwon and Lee (2020a) which follows a different approach by calculating a vulnerability score of the images for evaluation.

4 Methodology

For research methodology we followed a deep analysis approach to examine the best resources and practices that can be incorporated to achieve the desired setup for the research and results. The main concern and focus has always been automation of the entire process since the initial phase. The methodology phase began by searching about the open source resources that can easily be practised without any hassle. The field in which this research falls is cloud plus DevSecOps, where security is tried to be integrated in the DevOps. The field of research is the joint collaboration between operations, development and security. Automating the process in the DevSecOps is entirely essential as the security. Rangnau et al. (2020)

Jenkins and Gitlab CICD were the options for the pipeline which will be the implementation for the test bed of the docker image testing for the vulnerabilities. In this section we will first discuss how Gitlab was selected as a better choice and CI CD for the purpose of our research. Continuous Integration(CI) and Continuous delivery(CD) is one the most common and well known practices followed for the swift delivery of new features. In order to achieve the fast delivery all the stages required to deploy a safe software are done automatically in a pipeline. Traditional security management is not enough to support this fast delivery in a pipeline. To secure this pipeline and to secure the docker images we have built a pipeline with tools incorporated in the CICD pipeline. While researching about the docker container security we realised the need of automating the security testing of docker containers and how this area of research is important. Our research finds about the Gitlab CICD catalyst for all the informed decisions made while implementing the setup. The automation of the process will majorly unfold in two stages : Continuous Integration (CI) and Continuous Deployment (CD) :

Continuous Integration(CI): Integration of code from different sources like developers or different gitlab runners are integrated in this stage. Bugs, errors and security issues can be detected and fixed at an early stage. The docker images on which scanning would be performed will be built in this stage, once the build is successful the image will be pushed to gitlab private registry. This stage which is termed as jobs in teh gitlab will be triggered only after the code changes has been pushed to the Gitlab repo. Once the docker image is pulled, scanning tools will be integrated into the pipeline to scan the docker image against known vulnerabilities and CVEs. The entire process will be automated. Once the image is scanned and vulnerabilities are detected, evaluation based on

our selected parameters will be performed to analyse the results. The results and security reports generated will decide for the execution of the continuous deployment stage which is the next stage of the pipeline or the cluster environment.

Continuous Deployment(CD): Docker images once scanned in the Continuous Integration are deployed in this stage only if they pass the previous stage. Set of automated acceptance tests are also integrated in the deployment pipeline to verify if there is any regression due to system failure. This stage can also be used to identify if there are any errors that have occurred by variance in the runtime environment. In our setup we have automated all the process and testing however some manual testing can also be included before pipeline shifts to the next stage. When all the stages and jobs are passed the docker can be automatically deployed to the production environment. We are notified if any errors occur or the pipeline fails abruptly, in such cases the pipeline is also stopped automatically.

Another important question was docker images which needs to be researched for the purpose of research methodology. We built and scanned the docker images in the pipeline, the choice of docker container images was made consciously. Cern docker image cc7 and was used for the scanning Further details about the image configuration has been discussed in section 5. Once the docker image was selected we created a docker file in our CICD pipeline and incorporated all the tools required for carrying out research. We researched for Gitlab extension to add docker files and tool code to our pipeline to make our design setup function. 31 (2011) Once the code was refracted and working we scanned for the vulnerabilities on the images. The results were analysed. After the implication, we evaluated the result on the basis of extensive vulnerabilities scanned by both the tools Clair and Trivy detected vulnerabilities. We evaluated the results and finding of both the tools based on the vulnerabilities already present on the image, we also analysed how the tools are different for scanning vulnerability on the docker images. How the results of the tools differ and which tool might be preferable over the other based on the results drawn from the tools. Additionally we analysed the scanning results of two different images to test the accuracy of the design setup of the implementation.

5 Design Specification

The framework or the architecture we researched and set up for the purpose of our research work mainly revolves around the Gitlab CI CD that we had to build from scratch as per our requirements and suitable to meet the original research proposal. To lessen the burden of security concern we had to integrate vulnerability scanning tools Clair and Trivy into the CI CD pipeline and automating the entire process. The setup is expected to be fully automated, faster, low cost since all the resources are open source. Moreover the CI CD pipeline should be able to achieve the static analysis and complete automation for the developers. This section of the paper will look at some of the requirements necessary for the design and different components of the framework used. Below are the requirements for the Gitlab CI CD before any implementation of the security tools:

- **Requirement1 Build time**: Quick build time is one of the very initial requirements, ensuring that every static security scanning and build commit does not exceed 10 minutes for execution and can allow fast build fixes.

- **Requirement2 Parallel Pipeline jobs**: CI CD pipeline should be able to run all the different tests in parallel separate jobs, the test can be Unit, functional,

regression or security test at multiple abstraction levels. Parallel running if jobs can ensure a speedy execution.

- **Requirement3 Multi-versions test**: The pipeline should be able to smoothly test multiple versions of branches and commits.
- **Requirement4 Testing every commit**: Every commit made to the pipeline should be triggering the pipeline process
- **Requirement5 Necessary building**: Buggy build and codes should be avoided. The pre-built images should not require frequent updating.
- **Requirement6 Elastic strategy**: The deployment should be fully supported by configuration methods provided by the pipeline. Some systems can be deployed by some vulnerabilities where build failure is set to true or pass; however systems like docker images can not be deployed with vulnerabilities. The CI CD pipeline job failure state can be customised as per need. Other strategies that require attention are selecting the tests and at which stage of the CI CD process they need to be executed.
- **Requirement7 Vulnerability and Exposures reporting**: The pipelines should not just terminate or fail the job but also provide meaningful, human read errors that could be fixed.
- **Requirement8 Flexible testing and scanning**: The pipeline should be able to integrate the security testing tools easily as per the application of containers.

Once the requirements discussed above are taken care of, we will discuss docker and the docker container images along with the tool selection. The integration of the scanning tool in our CICD pipeline is also explained in the later part.

Docker: Docker containers, an open source technology based on the foundations of advanced container engineers. Docker was developed by DotCloud in 2013. Container technology overcame some of the limitations and challenges faced in virtualization. Containers use Linux kernel features, Cgroups and Namespaces to run independently. Initially Docke was based on Linux Container isolation(LXC). Kwon and Lee (2020a) The approach followed in the architecture consisting of client and servers are loosely coupled. All the components of docker being independent makes it a popular choice. Containers are nothing but a running instance which are termed as docker images. These docker images build the containers of docker, images are created as per the requirement and usage. Once an image is created it is uploaded to a trusted docker repository where anyone can easily access the image hasslefree.

Docker Images Deployment process: As discussed earlier docker containers are built on the basis of their images. Docker image is nothing but a qualitative package containing all the files , applications, middleware, OS, library and network configuration. The built image is uploaded to image repo, which can be used by users. Repositories can either be public or private. If an image is uploaded without any parameters through a docker pull image: name tag command then by default it is uploaded to Docker Hub. Same can be uploaded to private just by providing a private repository information in command. Diagram shows the process of the docker image being deployed after being built.Kwon and Lee (2020a) The same image can be created as a container just run command, after this command when performed through the docker client. Docker makes

a copy of the image, adding a layer on top of the base docker image to create the container. User sees an integrated view of the container as a single file system which is a docker image structure with multiple layers in it. Tasks that are performed by users such as apt-get install/upgrade are recorded in the container layer.

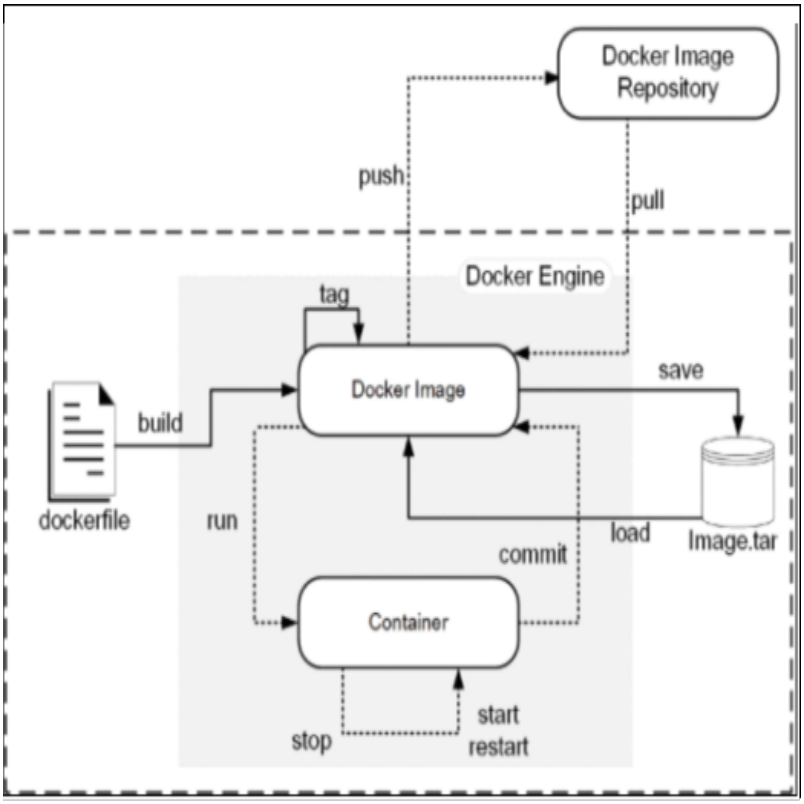

Figure 1: **Docker container workflow Kwon and Lee (2020b)**

Integration of Security scanning tool in CICD Pipeline: This section is an overview of the approach followed for the integration of the scanning tools in the Pipeline along with the requirements for a successful integration. The requirement for the security scanning tools in an iterative process, for initial the basic requirement was to create a basic initial version of the Gitlab CICD pipeline using .yml file.Rangnau et al. (2020) The detailed description of the implementation and design architecture is discussed in 7 The requirements for the pipeline to function and the integration of tools are discussed in the same section. The two security scanning and testing tools are decided for the final implementation depending on the requirements of the research. Furthermore details of the tools selected and their setup is discussed in 7. After the implementation of the tools it was time to test the execution of the pipeline. The evaluation, accuracy and testing outcomes and results were also the point of concern.

Tool selection: The tools that we used in our research were based on a few requirements of our research. The key being CLI(Command line interface), in order to integrate our pipeline and for the entire process being automated we looked for CLI based tools and selected which were open source and can easily be used. The first tool is : **Trivy:** Trivy is a versatile open source tool from Aqua Security 24 (2019) for scanning docker images in several OS packages such as Alpine, openSUSE leap, CentOS and Oracle Linux etc. Aqua Security acquired Trivy in 2019, later they became responsible for developing and maintaining it. Trivy can easily be integrated in Continuous Integration (CI) like

Gitlab, Travis, CircleCI, Jenkins and Github actions. It is also available as an extension for IDEs such as jetbrains and vim. There are no prerequisites to download libraries or other to make it work which makes its installation easier. Trivy has also the feature of scanning in application dependencies such as Bundler, npm, yarn and Pipenv. There has been several reasons to select Trivy as one of the tools which are discussed below:

Ease of use: Trivy is easy and simple in terms of usability; it does not require any launching of a database.
Pipeline integration: Trivy can easily be integrated with continuous integration pipelines such as Gitlab, Jenkins, CircleCI, and TravisCI which makes Trivy a suitable choice for a research project like ours. Trivy requires only an image or artefact to be specified to start and is simple. The scanning is fast as well and scans within seconds. Unlike other tools scanning time is faster in Trivy and does not require any maintenance.
In-Depth Report: Like Clair Trivy also provides an extensive report about the CVE scanned by the tools and provides the basic information about the found CVEs.
Features: Trivy was found to have scanned extensive and comprehensive vulnerabilities in comparison to other scanners available. Trivy also has multiple OS packages source to update the CVE vulnerability database such as CentOS(package image used in our project), Ubuntu, Red Hat Enterprise Linux, Universal base image, Oracle Linux, etc. Trivy also supports multiple application dependencies. The accuracy rate of the result by Trivy was also higher, especially with CentOS package images. One major factor to choose Trivy as the another scanning tool in our research project was the lack of literatures on the tool. We researched thoroughly but could not find any solid specific evidence of Trivy tool evaluation in a CICD pipeline through research papers or academic literature Trivy is just not limited to Containers but it can also target:

- AWS
- Kubernetes
- Virtual machines
- File system
- Git repository

Trivy can help to find not just containers vulnerabilities but also:

- Misconfiguration and IaC issues
- Sensitive secrets
- Common vulnerabilities and exposures (CVEs)
- Supply chain security (SBOM)

Integration of the tool in our CI CD pipeline and explanation of the artefact is explained in the section 6.

Clair: Clair is an open source project which provides tools for the monitoring of the containers security through static scanning of the docker container images. 26 (2019) While the user is working with containers they are also working with a part of the OS. It is essential to understand the vulnerabilities present in the libraries on the containers.

Vulnerabilities are continuously imported from known sources into the docker images in order to generate new vulnerabilities that pose a security threat to the docker containers. Clair scanning works by frequently updating its vulnerability data from different sources and exposes API for the users to invoke and perform scans on the containers. Clair scans and provides notifications based on the database from Common vulnerabilities and Exposures(CVEs) and similar databases from Ubuntu, Debian and RedHat.26 (2019) Clair supports multiple programming languages such as Python.There has been several reasons to select Clair as one of the tools which are discussed below:
Ease of Use: Clair requires the launching of the DB and service after the service is launched the tools are ready to work just by a command however tools like Anchore additional launching of engineer containers and separately require editing of config scripts to be connected to CLI Services. Moreover, the integration of tools like Anchore is not as convenient as Clair. Clair requires few commands to run its service which further makes it an optimum choice of tools, unlike others that require tons of commands to work. Integration of Clair is relatively easy and can easily be launched in old containers in the pipeline. **Pipeline integration:** Clair requires a single docker-compose file which launches a DB, and starts scanning just by a single command. The clocked time of launching Clair in the pipeline and making it run is relatively less and clocked at an average of 0:45 seconds. It can spin up fast as compared to other tools such as Anchore or Docker bench which require rewriting of the configuration file and requires a few different environment variables and several images for launching. **In-Depth Report:**Clair gives an extensive in-depth report of the found CVEs with the list of packages and links to them. **Features:** Clair can be easily integrated with Jenkins, Gitlab, and Kubernetes. Clair tool also has GUI, policies can also be applied which are less complicated to use, unlike other tools. Another major factor that makes Clair our choice of tool for scanning is its built-in drivers. Clair works with extensive CVE sources such as Ubuntu CVE tracker, NIST, and Debian. The Clair vulnerability database gets regularly updated with regard to several built-in drivers such as NIST, Red Hat security data, Debian security bug tracker, Alpine SecDB, etc. Since the CVE data sources are regularly updated in sync with the Clair vulnerability database the tool scans updated CVEs as well.

Integration of the Clair tool in the CICD pipeline and the challenges faced are discussed in the section 6. The figure below captures the entire design :

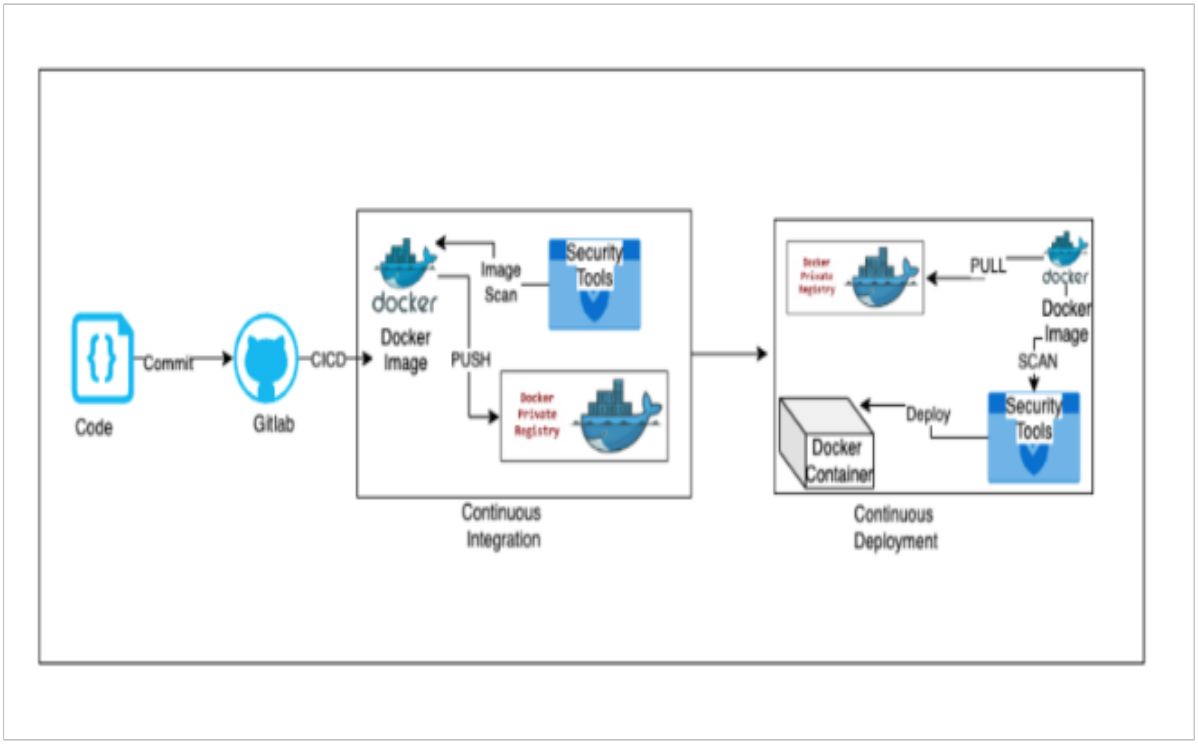

Figure 2: **Continuous Integration and Deployment process**

6 Implementation

In this section the entire implementation of the research design will be explained, all the steps taken to make the implementation will also be captured in detail.

Configuration	
Gitlab Enterprise edition	15.7.0
Trivy tool	Version : v0.35.0
Clair tool	Version : v8.0
Docker Image 1	cern/cc7-base
Docker Image 2	centos/httpd-24-centos7

Table 2: Configuration

Implementation of Gitlab CICD Pipeline:

Some prerequisites for creating a Gitlab pipeline are to have a project created in the Gitlab after registering an account where we can use the CICD pipeline. Pipelines are essential for continuous integration, development, delivery and deployment, it compromise of several components but the ones which are essential to make a pipeline function are:

- **Pipeline Jobs**: Pipeline jobs compile and test the code written. Jobs are responsible for defining the actual goal we are trying to achieve through the pipeline.

- **Pipeline Stages**: Stages define the order in which the jobs are supposed to run. 32 (2016) In order to run the jobs in the pipeline we need to configure the Gitlab runner.33 (2019) Runners is an application which works with Gitlab to make its jobs run, no job can run without a runner which makes it one of the essential steps. Configuring the Gitlab runner is essential, users get an option to install the runner on their local machine or to use a shared runner from the browser itself which does not require any additional installation. For our project we worked with both shared as well as locally installed runners. However by the end of implementation the Shared runner from the browser was successful enough to make every job run in our pipeline. Once the runner was available we created a .gitlab.ci.yml file in our root repository. 34 (2014) The stages that are defined in the Gitlab yml file are:

- **Build**: This stage is responsible for building our docker images.

- **Test**: This stage would be responsible for testing, scanning our images with the integrated tools. After the pipeline and runner are ready, it's time to incorporate the docker image through its docker file into the pipeline. In the repository files we created a docker file and committed all the changes. Later we wrote the code for pulling the docker images from the repository into the CICD pipeline. For scanning we added the code of Trivy and Cair tool in the testing stage of the pipeline through the CICD pipeline editor . After adding the code of both the tools, it was reflected for errors and then we moved onto running the pipeline. We ran the pipeline from the CICD tab. The pipeline reflected a passed status and the Jobs : Container build and Testing were successful, we critically analysed and evaluated the findings of the tools. The entire implementation led to an automated pipeline which worked on its own just by a click after all the requirements and configurations were successfully met.

7 Evaluation

This section provides a logical analysis on the result of the research carried out. We have discussed how docker images are vulnerable and need to be scanned before being deployed. The evaluation section of our design and implementation will provide the final results and comparison using appropriate parameters to support how our research has been able to successfully achieve the research question and the initial proposal. For the purpose of this evaluation we will compare how the two tools Clair and Trivy which are incorporated in our automated CI CD pipeline as discussed in sections 4 and 6 scanned the vulnerability on the vulnerable docker images which we have used for scanning. Both the tools are incorporated in the pipeline and were used to scan the vulnerabilities on the two images to draw an evaluation comparison and also to analyse how the two tools are scanning on different images.

- **Image 1** : cern/cc7-base

- **Image 2** : centos/httpd-24-centos7

We evaluated how Trivy and Clair provided different results in terms of the type of vulnerabilities they scanned depending on their severity level. To our surprise, the Trivy tool gave commendable results in terms of the number of vulnerabilities and was able to scan for a good number of vulnerabilities on our base image with some Major, Minor and Information severity vulnerabilities. Clair tool also detected vulnerabilities but were low in numbers as compared to the Trivy on the same docker images, however Clair found some critical vulnerabilities as well. For the evaluation we compared:

- The vulnerabilities discovered by the Trivy and Clair tool from the official repository of the packages in the docker image to check whether the design implementation is successful to find the official known vulnerabilities mentioned in the documentation. 26 (2019)
- We analysed the vulnerabilities scanned by the two tools on the basis of their severity on both the images.
- We also analysed how the two tools Trivy and Clair's results were different from each other on the same image in order to find the vulnerabilities.
- We also analysed our design framework on 2 images to evaluate how the implementation is working for different images.

The docker image file on which the evaluation is performed:

Image 1 : cern/cc7-base Image 2 : centos/httpd-24-centos7

The main findings are from the testing stage, once the container build is successful the pipeline CI moves onto the next job of testing. Below are the results from Trivy tool on the docker image :


```
10022    },
10023    {
10024      "type": "issue",
10025      "check_name": "container_scanning",
10026      "categories": [
10027        "Security"
10028      ],
10029      "description": "CVE-2017-6350 - vim-minimal - 2:7.4.629-8.el7_9 - vim: Integer overflow at a
    n unserialize_uep memory allocation site",
10030      "fingerprint": "1c8284651de9b3b66ff374515544c54244817240",
10031      "content": "An integer overflow at an unserialize_uep memory allocation site would occur for
    vim before patch 8.0.0378, if it does not properly validate values for tree length when reading a c
    orrupted undo file, which may lead to resultant buffer overflows.",
10032      "severity": "info",
10033      "location": {
10034        "path": "trivy-ci-test:786dc4bce85834a3d0a28f17b82ac502d8e59501 (centos 7.9.2009)",
10035        "lines": {
10036          "begin": 0
10037        }
10038      }
10039    },
10040    {
10041      "type": "issue",
10042      "check_name": "container_scanning",
10043      "categories": [
10044        "Security"
10045      ],
10046      "description": "CVE-2021-3903 - vim-minimal - 2:7.4.629-8.el7_9 - vim: heap-based buffer ove
    rflow vulnerability",
10047      "fingerprint": "cece352eb7acf67c7f2cde622e3b9f192fecd5c5",
```


Figure 3: **Test Results - Trivy**

Above screenshot captures just a section of results with found vulnerability, with the Trivy tool we could successfully detect a total of 1037 vulnerabilities in which few were Major, Minor and Informative severity known vulnerabilities that were present on the Cern/cc7 docker images. On Image 2 centos/httpd-24-centos7 Trivy detected a total of 1174 vulnerabilities in which few were Major, Minor and Informative severity.

Graph1. Shows the total vulnerabilities scanned by Trivy on Image 1 : cern/cc7-base

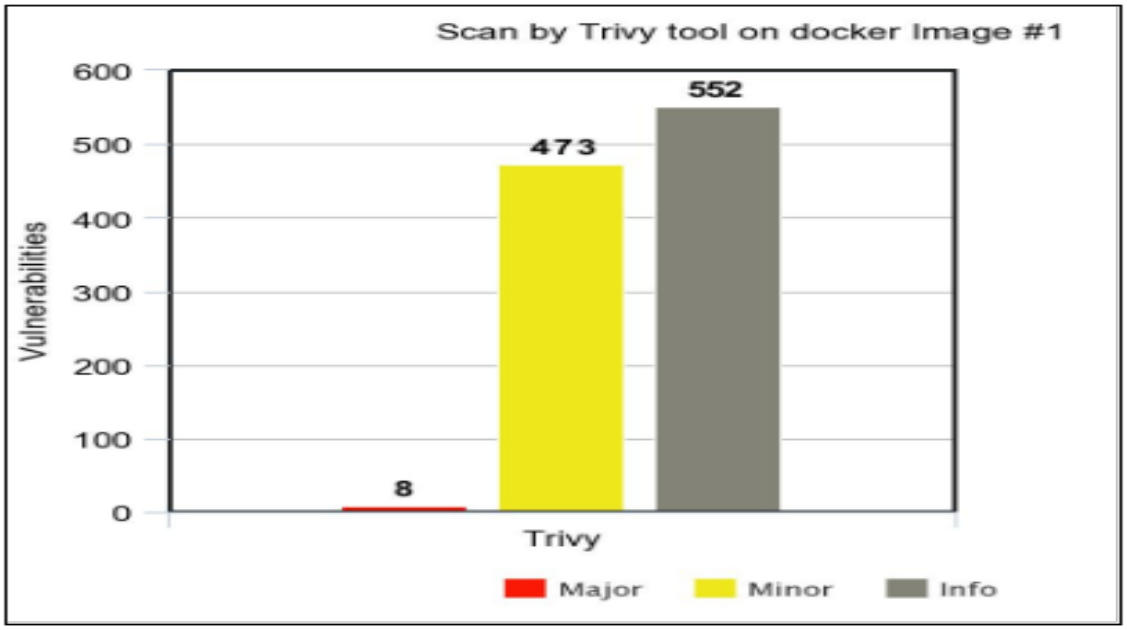

Figure 4: **Graph 1**

Graph2. Shows the results of scan by Trivy on Image 2 : centos/httpd-24-centos7

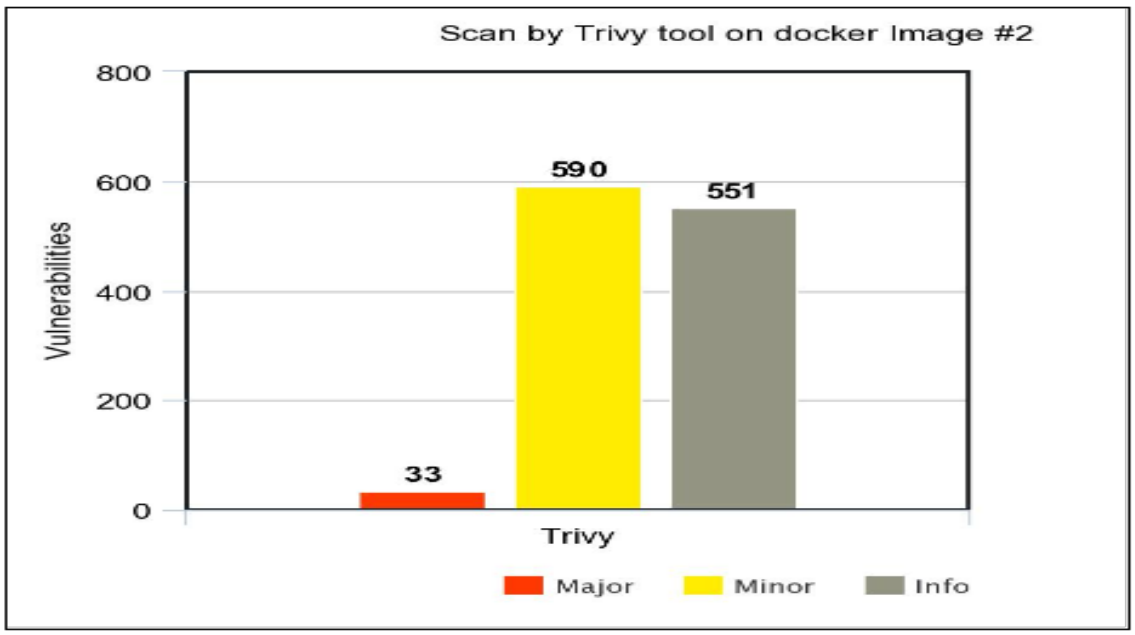

Figure 5: **Graph 2**

The Graph3. Analysis total vulnerabilities scanned by Trivy on Image 1 : cern/cc7-base and Image 2 : centos/httpd-24-centos7

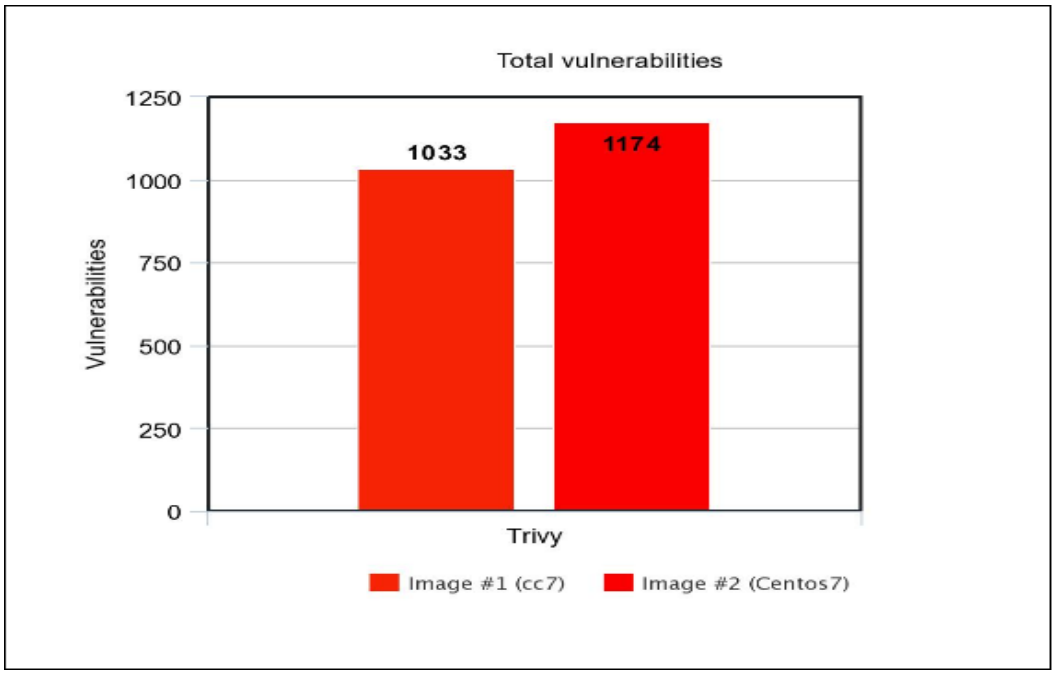

Figure 6: **Graph 3**

Clair Tool: Clair tool managed to detect vulnerabilities on both docker images, though the results of scan on the second image were very low in numbers as compared to Trivy tool. One thing to note here is in Clair, when the pipeline detected vulnerabilities on the images, the running job got terminated with an error. On Image 1 cern/cc7 Clair detected a total of 3 vulnerabilities in which of High Severity, on Image 2 centos/httpd-24-centos7. Clair detected a total of 80 vulnerabilities, in which few were Critical, high, medium and low severity.

The Graph4. Shows the total vulnerabilities scanned by Clair on Image 1 : cern/cc7-base

The Graph5. Shows the results of scan by Clair on Image 2 : centos/httpd-24-centos7

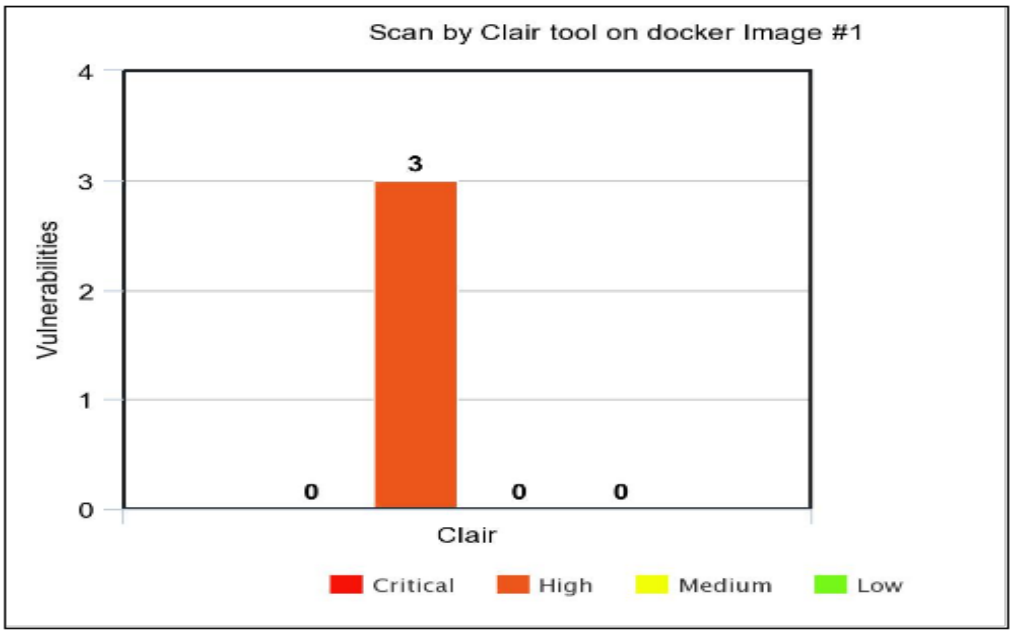

Figure 7: **Graph 4**

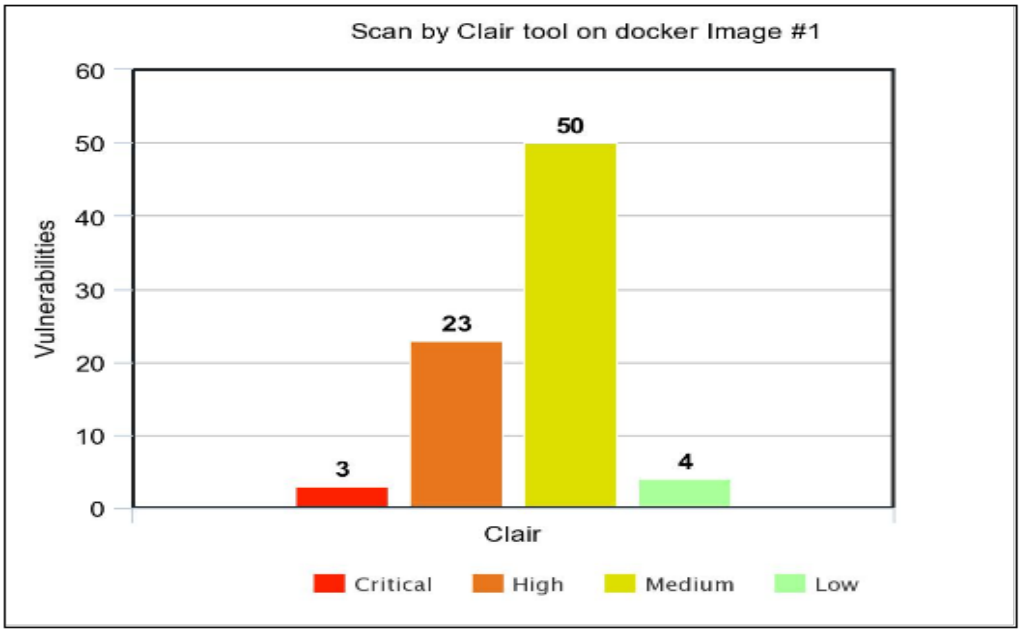

Figure 8: **Graph 5**

The Graph6. Analysis total vulnerabilities scanned by Clair on Image 1 : cern/cc7-base and Image 2 : centos/httpd-24-centos7

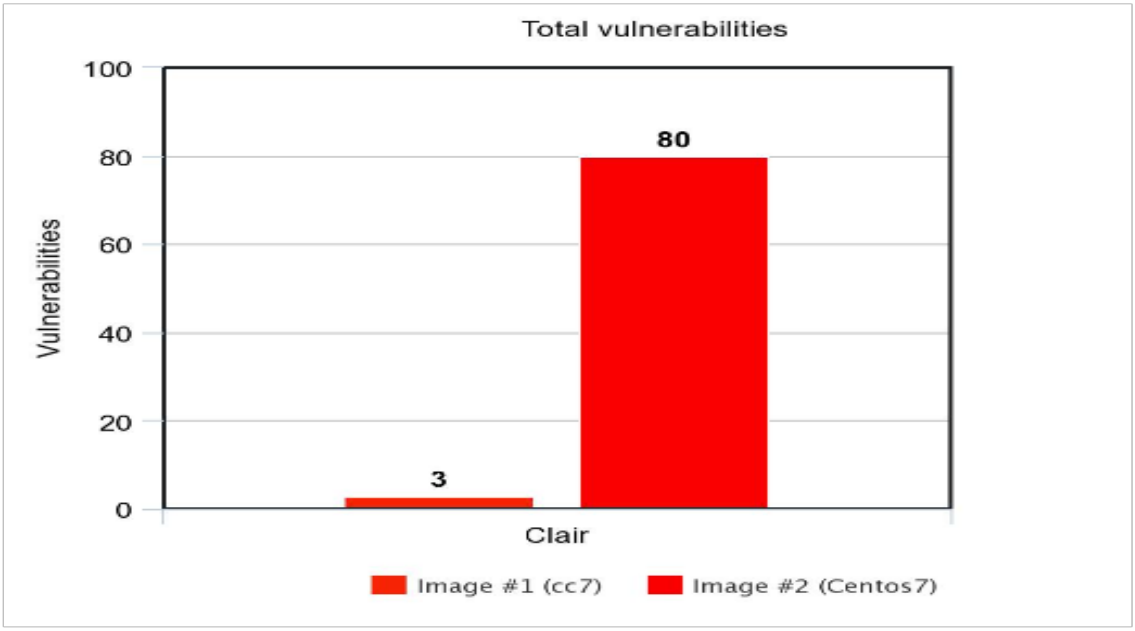

Figure 9: **Graph 6**

Both the tools scanned different number of vulnerabilities on both the images, Trivy discovered more vulnerabilities then Calir however, surprisingly Clair discovered some critical vulnerabilities on Image 2 : centos/httpd-24-centos7 which Trivy could not discover, also Trivy did not detect much vulnerabilities of High severity on Image 1 : cern/cc7-base as Clair did.

The Graph7. Analyses the vulnerabilities detected by Clair and Trivy as per their Severity on the Image 1 : cern/cc7-base as Clair

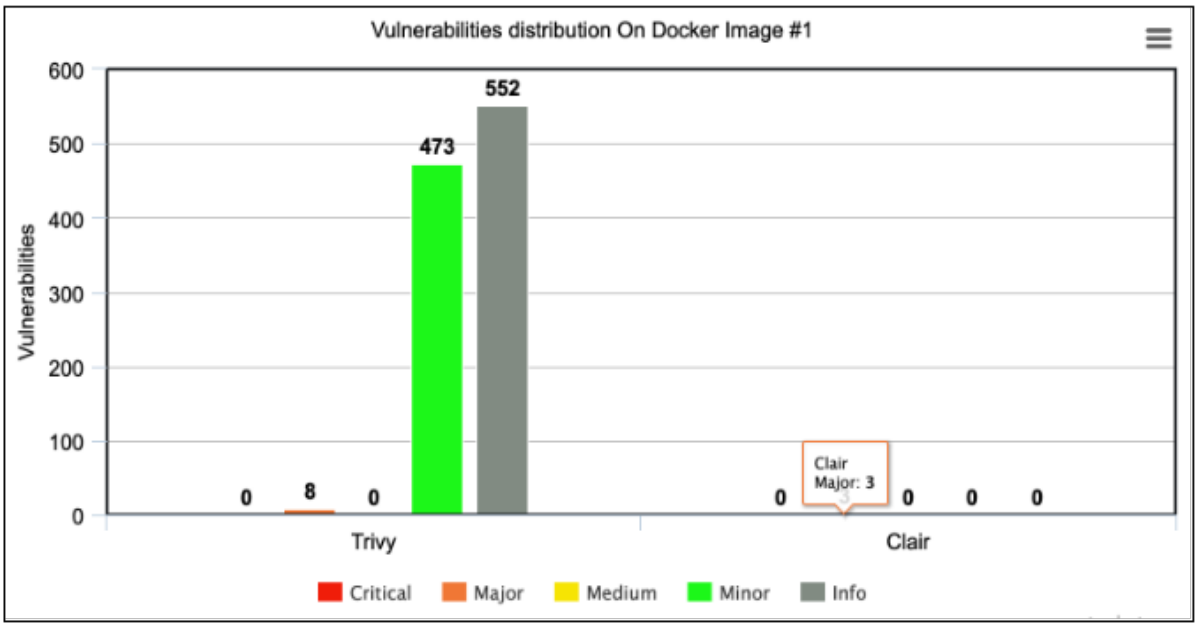

Figure 10: **Graph 7**

The Graph8. Analyses the vulnerabilities detected by Clair and Trivy as per their Severity on the Image 2 : centos/httpd-24-centos7

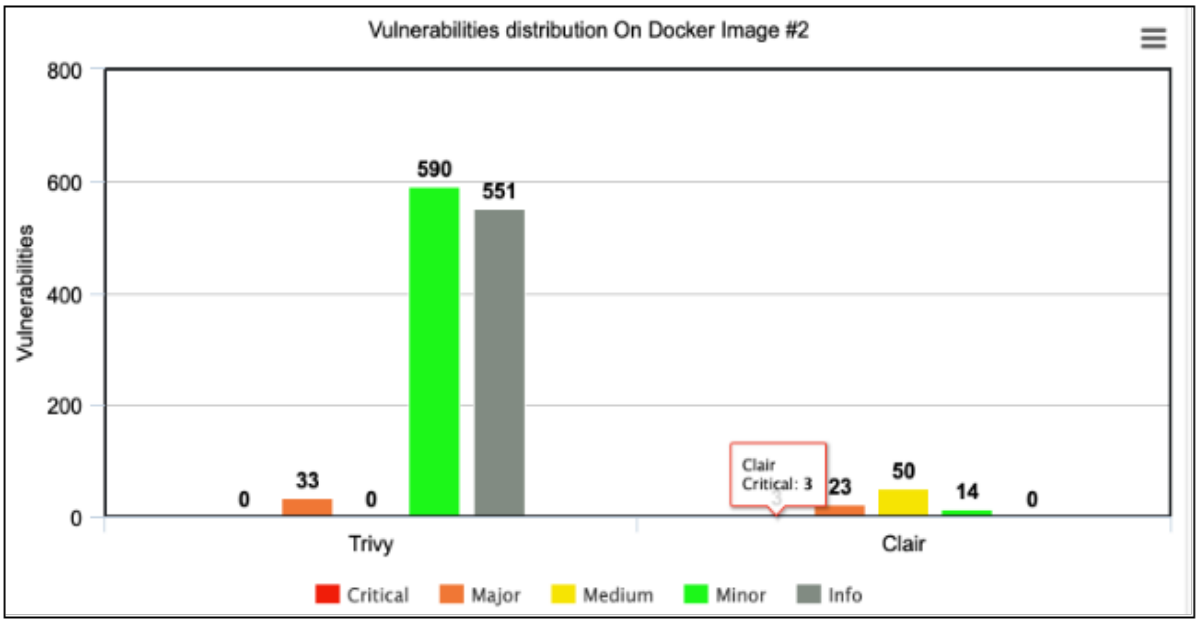

Figure 11: **Graph 8**

Analysis of the result by our design implementation: In the table below we have taken few of the vulnerabilities majorly of HIGH severity from scanning tools Clair and trivy, the packages present in the docker image 1(cern/cc7-base) are from redhat official repository 35 (2015) which is mentioned in the clair vulnerabilities reports and results as well 12 represents the information about the official location to check for the vulnerability source, along with the vulnerable package and CVE present in it. For 13 we have analysed few such vulnerabilities on both the tools from the official repository.

STATUS	CVE SEVERITY	PACKAGE NAME	PACKAGE VERSION	CVE DESCRIPTION
Unapproved	High RHSA-2022:8640	krb5-libs	1.15.1-54.el7_9	Kerberos is a network authentication system, which can improve the security of your network by eliminating the insecure practice of sending passwords over the network in unencrypted form. It allows clients and servers to authenticate to each other with the help of a trusted third party, the Kerberos key distribution center (KDC). Security Fix(es): * krb5: int rabilities in PAC parsing (CVE-2022-42898) For more details about the s including the impact, a CVSS score, acknowledgments, and other related information, refer to the CVE https://access.redhat.com/errata/RHSA-2022:8640

Figure 12: **Clair scanning results**

Vulnerability detected	Image CVE (cern/cc7-base)	Vulnerability Severity/impact	CVSS score	Clair	Trivy	Cern official repository	Vulnerability type
	CVE-2016-3191	Major	6.8	❌	✅	✅	**Buffer overflow**
	CVE-2015-8385	Major	6.8	❌	✅	✅	**DoS (denial of services)**
	CVE-2022-42898	Major	6.4	✅	✅	✅	**DoS (denial of services)**
	CVE-2016-1568	Major	4	❌	✅	✅	**Remote code execution**

Figure 13: **Examples of Vulnerabilities**

Apart from these vulnerabilities there are many more vulnerabilities which were detected by the Trivy and Clair tool as additional vulnerabilities. While doing the literature survey we found other research work which has used some similar tools in different settings and environments however to scan docker images. In the next section we will discuss some of the impacts of these major vulnerabilities detected through our design implementation. Most of the major attacks detected in the scanning can pose great security concern to the docker images, on researching deep it was found these issues are common in other docker images mainly Alpine 3.16 and docker kibana 6.8.16 Synk (2021) Therefore, reminding the research question again which is to explore docker security through an automated design implementation.

CVE-2015-8385 Impact Analysis: This vulnerability allows remote users to execute denial of services on the docker images.

CVE-2016-1568 Impact Analysis: This vulnerability is exploited by malicious actors who can execute arbitrary code on the host system over which docker image might be used.

CVE-2022-42898 Impact Analysis: This vulnerability can also execute a denial of services, and can also let applications crash. The similar vulnerability is detected in Alpine 3.16 docker image. Dotnet (2020)

8 Conclusion and Future Work

In this research paper we proposed a method to harden the security of vulnerable docker images through an automated pipeline, which can detect the vulnerabilities on the docker images. We successfully implemented the pipeline and executed the integration of the scanning tools to strengthen docker image security. The implementation automates the process of building, testing/ scanning the docker images, and giving the CVE reports which make it convenient for anyone to use the setup for scanning the docker images.

Docker images are a popular choice over virtual machines but they still face a lot of security challenges. The method we have used mainly consists of an automated CI CD pipeline which has scanning tools Clair and Trivy integrated in it to detect the vulner-

abilities present on the vulnerable docker images. Through the implementation setup we successfully managed to scan the docker images and detected the vulnerabilities present on both the images which validated that it can be easily used to scan other docker images as well. The results were imperative from both the scanning tools for Image 2 : centos/httpd-24-centos7, on which Clair tool detected some critical vulnerabilities as well. However on Image 1 : cern/cc7-base, Trivy gave better scanning and detection outcomes. This research contributed majorly in automating the process of detecting the docker vulnerabilities by Trivy and Clair. There is not much research work done using Trivy, hence this paper can be a great contribution to understanding the scanning of docker and the implementation, working of Trivy and successfully detected vulnerabilities in the 2 images used which had some critical, high, medium, and low vulnerabilities.

For the future we would like to incorporate a few other tools to improve the detection and also use multiple vulnerable docker images. We will also consider exploring dynamic scanning as well on the docker images, it could be something to explore and learn from. Also in this research we could not implement the dynamic analysis, which would be something to explore in future. Introducing machine learning techniques for detection and dynamic analysis can also be a greater extension of the work performed so far.

References

(2011). Gitlab ci/cd.

(2014). Tutorial: Create and run your first gitlab ci/cd pipeline.

(2015). Access to 24x7 support and knowledge.

(2016). Ci/cd pipelines.

(2019). Gitlab runner.

(2019). Trivy documentation.

(2019). What is clair?

Abhishek, M. K. and Rajeswara Rao, D. (2021). Framework to secure docker containers. In *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, pages 152–156.

Brady, K., Moon, S., Nguyen, T., and Coffman, J. (2020). Docker container security in cloud computing. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0975–0980.

Combe, T., Martin, A., and Di Pietro, R. (2016). To docker or not to docker: A security perspective. *IEEE Cloud Computing*, 3(5):54–62.

Dotnet (2020). Alpine 3.16 base image vulnerable to cve-2022-42898 issue 4229 dotnet/dotnet-docker.

Duarte, A. and Antunes, N. (2018). An empirical study of docker vulnerabilities and of static code analysis applicability. In *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*, pages 27–36.

Kwon, S. and Lee, J.-H. (2020a). Divds: Docker image vulnerability diagnostic system. *IEEE Access*, 8:42666–42673.

Kwon, S. and Lee, J.-H. (2020b). Divds: Docker image vulnerability diagnostic system. *IEEE Access*, 8:42666–42673.

Mahajan, V. B. and Mane, S. B. (2022). Detection, analysis and countermeasures for container based misconfiguration using docker and kubernetes. In *2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS)*, pages 1–6.

Manu, A. R., Patel, J. K., Akhtar, S., Agrawal, V. K., and Murthy, K. N. B. S. (2016). Docker container security via heuristics-based multilateral security-conceptual and pragmatic study. In *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pages 1–14.

of Computer Forensics, Z. J. I., Jian, Z., Forensics, I. o. C., of Computer Forensics, L. C. I., Chen, L., of Posts, C. M. A. Z. J. C. U., Publicati, T., of Posts, Z. J. C. U., and counts1Available for Download1Citation count21Downloads (cumulative)1, T. P. Y. P. (2017). A defense method against docker escape attack: Proceedings of the 2017 international conference on cryptography, security and privacy.

Pathirathna, P. P. W., Ayesha, V. A. I., Imihira, W. A. T., Wasala, W. M. J. C., Kodagoda, N., and Edirisinghe, E. A. T. D. (2017). Security testing as a service with docker containerization. In *2017 11th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, pages 1–7.

Pinnamaneni, J., S, N., and Honnavalli, P. (2022). Identifying vulnerabilities in docker image code using ml techniques. In *2022 2nd Asian Conference on Innovation in Technology (ASIANCON)*, pages 1–5.

Profile, T. G. A. V. (2016). 5 container security risks every company faces.

Rangnau, T., Buijtenen, R. v., Fransen, F., and Turkmen, F. (2020). Continuous security testing: A case study on integrating dynamic security testing tools in ci/cd pipelines. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 145–154.

Reeves, M., Tian, D. J., Bianchi, A., and Celik, Z. B. (2021). Towards improving container security by preventing runtime escapes. In *2021 IEEE Secure Development Conference (SecDev)*, pages 38–46.

Seals, A. T. and Seals, T. (2015). Poorly secured docker image comes under rapid attack.

Sengupta, R., Sai Prashanth, R. S., Pradhan, Y., Rajashekar, V., and Honnavalli, P. B. (2021). Metapod: Accessible hardening of docker containers for enhanced security. In *2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pages 01–06.

Shameem Ahamed, W. S., Zavarsky, P., and Swar, B. (2021). Security audit of docker container images in cloud architecture. In *2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)*, pages 202–207.

Singh, C., Gaba, N. S., Kaur, M., and Kaur, B. (2019). Comparison of different ci/cd tools integrated with cloud platform. In *2019 9th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, pages 7–12.

Synk (2021). Vulnerability report for docker kibana:6.8.16.

Tunde-Onadele, O., He, J., Dai, T., and Gu, X. (2019). A study on container vulnerability exploit detection. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pages 121–127.

University, R. S. N. C. S., Shu, R., University, N. C. S., University, X. G. N. C. S., Gu, X., University, W. E. N. C. S., Enck, W., University, A. S., Mnchen, T. U., Boston, U. o. M., and et al. (2017a). A study of security vulnerabilities on docker hub: Proceedings of the seventh acm on conference on data and application security and privacy.

University, R. S. N. C. S., Shu, R., University, N. C. S., University, X. G. N. C. S., Gu, X., University, W. E. N. C. S., Enck, W., University, A. S., Mnchen, T. U., Boston, U. o. M., and et al. (2017b). A study of security vulnerabilities on docker hub: Proceedings of the seventh acm on conference on data and application security and privacy.

Wenhao, J. and Zheng, L. (2020). Vulnerability analysis and security research of docker container. In *2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE)*, pages 354–357.

Yang, Y., Shen, W., Ruan, B., Liu, W., and Ren, K. (2021). Security challenges in the container cloud. In *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 137–145.