PROJECT REPORT

An Investigation into Container Security and the Effectiveness of Security Controls against Vulnerability Exploitation

Investigative Project

Jonathan Lee Hill

2021 / 2022

Authorship Declaration

I declare the following:

(1) that the material contained in this dissertation is the end result of my own work and that due acknowledgement has been given in the bibliography and references to ALL sources be they printed, electronic or personal.

(2) the Word Count of this Dissertation is 17,775.

(3) that unless this dissertation has been confirmed as confidential, I agree to an entire electronic copy or sections of the dissertation to being placed on the eLearning Portal (Blackboard), if deemed appropriate, to allow future students the opportunity to see examples of past dissertations. I understand that if displayed on eLearning Portal it would be made available for no longer than five years and that students would be able to print off copies or download.

(4) I agree to my dissertation being submitted to a plagiarism detection service, where it will be stored in a database and compared against work submitted from this or any other School or from other institutions using the service.

In the event of the service detecting a high degree of similarity between content within the service this will be reported back to my supervisor and second marker, who may decide to undertake further investigation that may ultimately lead to disciplinary actions, should instances of plagiarism be detected.

(5) I have read the Northumbria University/Engineering and Environment Policy Statement on Ethics in Research and Consultancy and I confirm that ethical issues have been considered, evaluated and appropriately addressed in this research.

SIGNED: Jonathan Lee Hill

Acknowledgements

Thank you to my supervisor, Rafay Ansari, for providing guidance, feedback, and support throughout this project. Rafay always ensured I was on the right track and pointed me in the right direction.

I would also like to thank Xiaomin Chen, Gerhard Fehringer, and Rafay for all the support during my time at university. Both lectures and seminars were taught exceptionally and made learning new techniques and skills enjoyable.

Finally, I would also like to thank my family for their support throughout this year. Working through covid has been hard for all of us, yet they were always there to support me and give me the determination to stay focused on my work.

ABSTRACT

By 2022 more than 75% of organisations are expected to be running containerised applications within production; this growing popularity of container images within cloud-native development has caused concern about the security of container image usage within organisational pipelines. Whilst more and more organisations are adopting containerised applications within production, there is a growing concern for the security of container images due to the risk of vulnerable images within production environments. Organisations that do not follow a security control framework within their CI/CD pipeline will risk system-wide compromises leading to further economic and ethical problems. Recent breaches in the field have shown catastrophic losses for organisations involved creating further concern over the security of container technology security. To mitigate against the risk of insecure container images, organisations must secure their CI/CD pipelines in order to secure container image development and deployment. Properly securing software pipelines can help towards mitigating security risks. Vulnerability assessment tools play an essential part in securing the pipeline and protecting organisational production environments. The vulnerability assessment tools are used in a typical pipeline's testing and merging stages to scan for common vulnerabilities and exploits. Within the build phase of a pipeline, static vulnerability assessment tools are used to scan images for CVEs. Tools such as Jfrog Xray provide organisations with dynamic analysis of container images by simulating production environments. These simulated environments are used to run tests on active containers to discover remote exploits that only appear when the container is active.

Previous studies have investigated the effectiveness of dynamic vulnerability assessment tools. However, there is a lack of awareness of the effectiveness of static vulnerability tools. This paper aims to provide an insight into the popular NIST security control framework and provide analysis, discussion, and evaluation of the effectiveness of popular open-source, static vulnerability assessment tools. A Virtual Machine is used to provide a Linux Ubuntu OS within the investigation. The three vulnerability assessment tools chosen for this study include Dagda, Grype and Trivy. All three tools are open-source and provide static analysis of container images through a Command Line Interface within the Linux VM. The tools within the study were used to run rigorous tests on twenty container images to scan for CVEs within both OS and non-OS libraries. The results from the test are used to provide an overall analysis of the effectiveness of each tool.

Based on the research and data from the container image tests, this study recommends Trivy for protection against attackers exploiting Common Vulnerabilities and Exploits. Trivy was the most effective at discovering CVEs within container images. The tool required fewer system resources than Grype and Dagda. It was found that Trivy discovered over thirty-two per cent more CVEs than the other tools tested during the investigation. This report also recommends that organisations introduce the NIST security control framework within their CI/CD

pipelines by implementing security control policies from the NIST 800-53 and NIST 800-190 special publications. Complying with the NIST security controls will help secure their organisational pipeline and mitigate against potential future security risks.

National security control standards are implemented within organisational pipelines to provide policies to secure the development and deployment of software. Drawing upon the current test results and existing NIST publications, this thesis recommends optimal security controls to develop, deploy and manage container images.

CONTENTS

1 INTRODUCTION

1.1 BACKGROUND

Container images are packages of software that work similarly to Virtual Machines that can be run within any environment. Containers are growing in popularity in cloud-native development over the use of Virtual Machines (VMs) because they are lightweight, have faster boot times and require less memory space. However, it is essential to understand the security of containers. With the wide adoption of container-based applications and systems alongside the introduction of DevSecOps, it has become apparent that container security is an essential aspect of container app development.

One of the major problems organisations face during software development is zero-day exploits. Zero-day exploits cause problems for software developers as they can result in data breaches. Organisations routinely update their software to help combat zero-day exploits and the ever-growing fear of newly found CVEs (Common Vulnerabilities and Exploits). The routine updates provide the software with new mitigations to protect against the newly found CVEs. Routinely updating software is automatically done through the organisation's CI/CD (Continuous Integration/Continuous Deployment) pipeline. Within the CI/CD pipeline, vulnerability assessment tools are used throughout all pipeline stages to ensure any new exploits and vulnerabilities are discovered. The introduction of vulnerability assessment tools within pipelines helps mitigate against the newly discovered CVEs and reduces the risk of falling victim to potentially fatal, zero-day exploits.

The reason container security and effective vulnerability assessment tools should be studied is that past research, such as the publication on Docker Container Security in Cloud Computing (Brady, et al., 2020), has used a dynamic analysis approach to assess container image vulnerabilities. Past research has shown a lack of awareness of the static analysis of container images. Therefore, this study aims to provide an insight into the effectiveness of static vulnerability assessment tools. Further research has also shown that containers provide less isolation than Virtual Machines due to kernel sharing and require additional security (Bélair, et al., 2019). The additional need for security within containers results from potential unauthorised root access to host servers from compromised containers. As root access gives users the highest level of privileges and access, proper container security must be implemented. Due to the potential system-wide compromises, containers must be adequately secured. Organisations can ensure strong container security by complying with a container security framework.

The National Institute of Standards and Technology (NIST) published an Application Container Security Guide under the 800-190 Special Publication. The executive summary of the publication stated that 'Organisations should adopt container-specific vulnerability management tools and process for images to prevent compromises.' (Souppaya, et al., 2017). These publications emphasise how important this study is to organisations adopting containerised applications within their production environments. The implementation of vulnerability management tools can be used to process images to prevent future system compromises. This paper aims to test the effectiveness of popular open-source vulnerability tools at processing container images.

Research shows that the Application Container Security Guide encourages organisations to use security tools that can enforce baseline requirements for vulnerability management and compliance prior to allowing an image to be run (Souppaya, et al., 2017). This research further stresses the importance of the study. The research states how it is essential that container images should be scanned for CVEs and security control compliance before being pushed to production environments.

Due to prior research demonstrating the importance of container security and an evident lack of research into the static analysis of container images, this paper aims to use static vulnerability assessment tools against container images alongside literature on appropriate security controls. This investigative study will use the results and research to provide organisations with security best practices and a security policy safeguarding the use of containers.

1.2 Aims & Objectives

1.2.1 Aim

This research aims to investigate container security controls from a popular security framework and compare and evaluate the effectiveness of open-source vulnerability assessment tools against container images. This investigative study aims to test a range of container images with multiple tools to provide a recommendation to organisations looking to secure their pipelines with the best available open-source static analysis tool depending on their organisational needs. Furthermore, the vulnerability assessment results will be used in conjunction with a security control framework analysis to provide secure organisational policies to provide better security within company software development.

1.2.2 Objectives

1. **Review previously published similar works**

This objective is to identify any issues with container image use in the past and provide literature showing the importance of containerised application use in organisational environments and pipelines.

2. **Establish the uses of containers in large-scale enterprise workloads**

This objective is to identify how containerised applications are utilised within large-scale workloads such as cloud-native environments. This objective aims to strengthen the foundation on the importance of secure container technology and how many sectors containerised applications affect.

3. **Research tools and policies that provide security to containers and infrastructure**

This objective aims to provide the study with a critical review of literature on popular open-source vulnerability assessment tools and a security control framework that specifies policies for organisations to strengthen their information systems. This objective gives the reader insight into a popular security control framework to show how relevant security controls can be implemented within organisations to secure their infrastructure and containerised application development and deployment. The research also lays the foundation of the study and further develops the project's narrative to show the importance of the roles of vulnerability assessment tools within organisational pipelines.

4. **Perform in-depth analysis of container security, including security controls and features**

This objective consists of investigating container security as a whole, including a detailed analysis on relevant security controls from a popular security control framework. This objective aims to use the analysis to stress the importance of the project approach. As vulnerability assessment tools are considered the last security step within container development before the images are pushed to repositories, this objective aims to flow onto the design stage where vulnerability assessment tools are set up for the investigation.

5. **Plan, set up and configure container vulnerability tools**

This objective aims to show the design section of the investigation and provide insight into the vulnerability assessment tools chosen and the setup and configuration. This objective also provides the study with a justification for the approach to the investigation and reviews alternative approaches. This objective also consists of the design challenges faced throughout the project's investigation.

6. **Test popular container images for OS and non-OS vulnerabilities**

This objective aims to collect quantitative data from multiple container images by running various images through three popular open-source vulnerability assessment tools. The data collection method consists of extracting the data from each vulnerability assessment and storing it within a spreadsheet for further analysis. This objective aims to provide insight into the effectiveness of the three vulnerability assessment tools by scanning for both operating system and non-operating system vulnerabilities within a variety of active and depreciated container images.

7. **Provide an in-depth analysis of the number of vulnerabilities in container images for each tool used.**

This objective aims to provide the study with a detailed presentation of the investigation's findings, including an in-depth analysis of the tool comparisons. This objective aims to use the analysis to provide the study with an overview of the results and provide insight into how the data presents each vulnerability assessment tool's effectiveness. This objective aims to provide a critical discussion of the work done during the investigation.

8. **Evaluate the effectiveness of vulnerability tools**

This objective provides the study with a discussion of the work done, including both research and practical work. This objective aims to use this discussion to evaluate the investigation and the effectiveness of vulnerability assessment tools. This evaluation will cover investigative research and the analysis of results to conclude how effective vulnerability assessment tools provide security for organisational pipelines. This evaluation will also cover any alternative approaches that could be made if the project was to be repeated.

9. **Propose a secure container policy that can be applied to safeguard the use of containers by enterprises.**

This objective aims to provide organisations with a structured, secure security policy to strengthen their infrastructure and information systems when working with containerised application deployment and development. This objective will use the investigative work conducted during the study, including research and practical work, to evaluate a secure policy that organisations can follow to safeguard the use of containers. This policy will include multiple security controls relevant to container security and best practices for securing containerised application use.

1.3 MOTIVATION

The motivation for this investigative study is the perceived lack of business awareness of the importance of container security and static vulnerability assessment tools. This research project is important because previous research has shown the effectiveness of dynamic analysis tools. However, there is a lack of study on static container image testing. Investigating static vulnerability assessment tools can help industry developers, and organisations better understand security best practices and the effectiveness of static image testing on container images. This study's investigative work provides insight into relevant security policies that can aid containerised application development. The results from the investigation should allow organisations and developers to understand the efficiency of different open-source static vulnerability assessment tools and the effects security controls can have on organisational pipelines. As a result, the reason and motivation behind this study were due to the lack of awareness in static image testing and the potential aid to people in the industry of cloud computing and containerised application development.

1.4 Approach

Chapter 2 includes the research and planning steps of the project. Chapter 2.1 covers a critical review of literature research on the importance of container security within organisational development pipelines. Following the literature review, Chapter 2.2 includes a detailed discussion on the use cases of container images. Chapter 2.3 presents further research into the importance of secure containers, discussing recent security breaches and the role of security control policies. Chapter 2.4 provides an in-depth analysis of the most popular security control framework used with container software development, including a comprehensive breakdown of relevant security controls. Using the previous chapter, Chapter 2.4 also provides insight into secure CI/CD pipeline practices and the role of vulnerability tools within the lifecycle of a container.

Chapter 3 includes the discussion of the design, method and results. Chapter 3.1 involves a thorough discussion of the vulnerability assessment tools used within the study and a detailed justification of the tools used. Furthermore, Chapter 3.2 discusses the vulnerability assessment tool setup, including the collection of container images, the Virtual Machine configuration, and the vulnerability assessment tool configuration. After discussing the configuration of the tools, Chapter 3.2 also covers the design challenges faced within the planned approach. After covering the problems faced within the project design, Chapter 3.3 also includes a critical discussion of the container image testing process. After discussing the container image testing process, Chapter 3.4 presents the results found from each tool and provides a detailed comparison.

Finally, Chapter 4 concludes the investigation by providing a discussion and evaluation of the findings and the project process. Chapter 4 also includes a conclusion covering the entire project and recommendations for developers, organisations and further research in the field.

2 Research and Planning

2.1 Literature Review

Recent work on the current issues and challenges of container security has shown the importance of container security and the possible attacks and vulnerabilities—the report proposed future research directions in the hopes of spawning further research in the area. (Sultan, et al., 2019). It was mentioned that further research into the usability of vulnerability assessment tools would provide tremendous help to developers and companies. Previous work has also shown that security controls must be introduced to secure the use of container-based virtualisation (National Institute of Standards and Technology, 2017). In September 2017, NIST published the first guide on application container security. This publication discussed security threats, recommendations, and countermeasures to safeguard the use of application containers.

This paper provides an insight into recent security breaches involving container images and aims to use NIST publications to form a basis for the importance of container image security and framework controls. This paper also aims to investigate the steps organisations must follow to secure their CI/CD pipelines.

The idea for this project came from an interest in exploring a stronger alternative to Virtual Machines within organisational production environments. After conducting research into the area, an upcoming popular alternative to Virtual Machines was discovered that companies are adopting within their software development.

This project will be helpful to developers and organisations who are using containers as a lightweight alternative to Virtual Machines. This project aims to provide an in-depth analysis of container security and explore the effectiveness of the current security controls.

Within the study, multiple vulnerability assessment tools will be used to provide an insight into the effectiveness of each tool and show the risks of online docker images. Companies and organisations that do not implement security controls and vulnerability assessment tools within their pipelines will be at significant risk of security breaches and possible system-wide compromises. Organisations that lack security control compliance are at significant risk of confidential data breaches. Organisations must follow security control frameworks such as the NIST 800-53 and NIST 800-190 to mitigate potential security risks and reduce the impact of potential breaches.

2.2 USE OF CONTAINERS

In cloud-native development, containers have grown in popularity within organisations as an alternative to Virtual Machines. Containers usually are megabytes in size and are an extremely light substitute for Virtual Machines. This is because containers only virtualise the applications and libraries instead of the whole computer; in some cases, containers can also share libraries. Each container shares the host's operating system kernel, binaries, and libraries. As containers share operating system resources, servers can run up to three times the amount of applications over Virtual Machines.

Research has shown that by 2022, more than 75% of global organisations are expected to be running containerised applications within production, up from less than 30% in 2020 (Gartner, 2020). This research shows the importance of container technology security. As the global containerised application use rises, the demand for container security has grown astronomically. As more and more organisations adopt containerised applications, they must follow effective security control policies to protect their data and information systems.

2.2.1 Cloud Network

The growth in enterprise adoption of containers shows the appeal of cloud-native development. Cloud container usage provides significant benefits over conventional Virtual Machines, such as scalability and responsiveness.

2.2.2 Microservices

Due to containers being lightweight, they are well suited for applications that use a microservice architecture. The use of containers within a microservices framework will allow for a highly scalable system whilst allowing for the CI/CD method for application distribution. Companies like Amazon and Netflix have already re-architected monolithic applications into microservices applications (AVI Networks, 2022).

2.2.3 DevOps

The introduction of DevOps practices helps organisations increase their ability to deploy software and services faster than traditional methods. Past research has shown that if a vulnerability is used to exploit a container to a high enough degree, then the other containers and potentially the host itself can be compromised (Binnie, 2018). This research shows how important DevOps is within containerised environments. As multiple containers running on a singular host share the same kernel, a system-wide compromise can happen due to a single high severity level exploit. Due to the dangers of high-level exploits, DevOps practices are put in place within the lifecycle of containers to help mitigate against security risks. Due to the popularity of the microservice architecture, DevOps teams use containers to deploy services and scale their infrastructure using continuous integration/deployment (CI/CD) tools. CI/CD tools such as Jenkins and GoCD offer developers freedom over the use of traditional routes of software deployment.

The automation and monitoring that CI/CD tools provide allow developers to push newly updated software into the environment faster than traditional methods. Due to more and more exploits being found within commonly used software libraries, containerised applications must be updated frequently to keep up to date with new CVEs. Updating software to keep up to date with new CVEs is made a lot easier with the automation of a CI/CD pipeline. When new updates are made to an image, it can be pushed through the pipeline and released faster than traditional methods. Introducing DevOps within a CI/CD pipeline allows for real-time risk management as code can be tested and deployed faster than a traditional software lifecycle. Companies that introduce DevOps practices benefit from faster software delivery, improved communication and a more stable working environment. DevOps organisations can introduce automation to their software developments, resulting in a smoother deployment process that can save time. The introduction of automation can introduce higher productivity within the company and increase product quality and deployment success rate. Recent research has shown that the adoption of DevOps is growing rapidly, as the worldwide DevOps software market is forecasted to reach $6.6 billion in 2022 (AppDynamics, 2022). The ever-growing popularity of

DevOps practices is because its approach to software deployment enables faster deployment of new software and easier maintenance of existing deployments.

2.2.4 Multi-cloud environments

As container images are a standard software package, each container runs as a separate process that allows for scalability. The standardisation of code makes it easy for organisations and enterprises to quickly build and run applications on multiple environments such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform and easily migrate across platforms with no changes made to the software.

2.2.5 Application modernisation

Due to the multiple benefits of containers, it may be in an organisation's best interest to containerise their applications and move them to a cloud environment. The reason behind application modernisation is that containers allow faster deployment, patching and scaling. These efforts to speed up the DevOps cycles can also provide developers with more time to spend on other important jobs.

2.2.6 Enterprise Workloads

The introduction and use of containerised applications can help organisations optimise resource use and scalability. Due to containers lightweight applications, they are a great way for organisations to rapidly scale to customers needs without needs for additional infrastructure.

2.3 Importance of Container Security

2.3.1 Breaches

Previous work has shown that there have been multiple cases of security breaches within cloud computing and container platforms. In the 2018 report on the proceedings of the international conference on cloud computing, it was noted that the challenges of information security increase exponentially when a company adopts a cloud-based system over a more conventional distributed network system (Duncan, et al., 2018). It is essential to understand that this increase in difficulty can cause organisations to suffer.

Companies must adhere to the General Data Protection Regulation (GDPR) when adopting a cloud-based system. This regulation is the prominent cloud forensic problem companies face when adopting cloud-based systems and, if not followed correctly, can lead to system-wide compromises. Research shows that in 2017, Yahoo had a system-wide compromise, and all of its 3 billion accounts were breached (Khandelwal, 2017). Research has also shown that in 2019 the cloud-based Docker Hub Repository suffered a data breach that affected over 190,000 users (Trend Micro, 2019), which is very concerning since recent research has shown that if any company struggles with the forensic problem of their GDPR implementation, it will cause such a 'fundamental' problem that the company would not be able to comply with the regulation (Duncan, et al., 2018). This research shows how a lack of compliance with security regulations can cause significant data breaches and lead to substantial economical backsets for companies involved.

2.3.2 Pipeline Security

With the increasing threat of vulnerabilities and breaches, organisations need to secure their container pipelines and applications. Organisations must implement an automated pipeline to quickly update containerised applications when vulnerabilities/exploits are discovered and allow for fast responses to security incidents. As prior research has shown that an insecure pipeline inevitably leads to an insecure application, organisations must implement security controls within their pipeline to provide more security for containerised application development (Sysdig, 2022). Organisations must deliver secure pipeline security within containerised application development to prevent data breaches and potentially system-wide compromises. To increase organisational pipeline security, security frameworks must be introduced to provide controls to secure the lifecycle of containerised application use.

2.3.3 Secure Deployment Environments

It is crucial that organisations provide security controls for container deployment environments and infrastructure to prevent vulnerabilities and exploits within production environments. Many exploits only occur during the runtime of an application. Organisations must provide security towards their deployment environments to detect and prevent hidden vulnerabilities and exploits within container images.

2.3.4 Role of Security Controls

The role of security controls within a container's development lifecycle is to secure every step of the development of a container. Security controls are put in place within an organisation's pipeline to ensure proper security regulations and standards are being met. Without implementing a security control framework within an organisation's pipeline, security implementations and risk mitigations can be easily missed, leading to possible company data breaches once container images enter a production environment. Research has shown that over half of the 4 million images on the Docker Hub Repository have been found to contain critical vulnerabilities (Barua, 2020) This research is concerning as prior research has shown that the Docker Hub is one of the most popular container image repositories, and this research shows that over half of the images within the repository contain critical severity level vulnerabilities. Without implementing security controls within organisational environments, companies can be at risk of high-level vulnerabilities. This research shows how vital security control implementation is and how the lack of compliance with security controls could leave organisations open to high-level vulnerabilities and exploits.

2.4 Container Analysis

2.4.1 Comparison to Virtual Machines

The introduction of containerised applications has been a massive benefit to organisations due to their many advantages over traditional Virtual Machines. The diagram below shows how containerised applications are virtualised over conventional Virtual Machines.

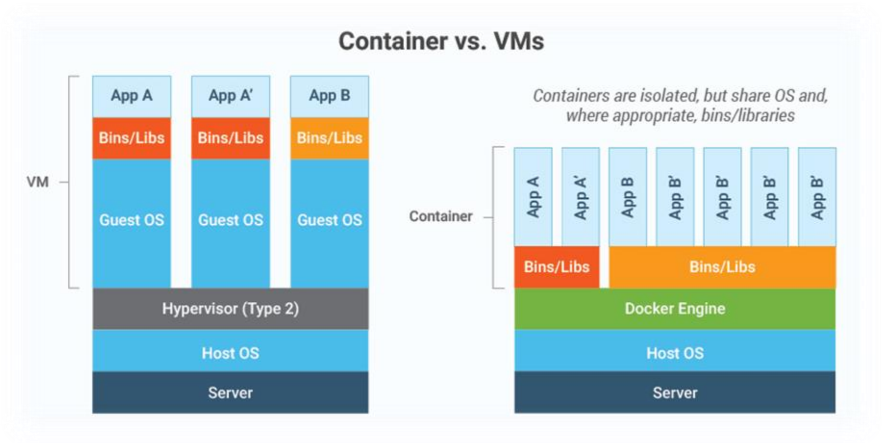


Figure 1, Diagram displaying the difference in virtualisation between Virtual Machines and containers (Warrier, 2020).

The figure above demonstrates how containers are lightweight in relation to Virtual Machines. The primary advantage containers have over Virtual Machines is scalability. Although Virtual Machines are ideal for 'lift-and-shift' migrations, the cost of scaling up Virtual Machines can increase exponentially due to their demand for resources. The fast deployment and scalability of containers are why they are taking over organisational application workloads. As seen in Figure 1, containerised applications are isolated however, they share the operating system and can have the ability to share bins/libraries. This results in containers having a lower resource footprint, limiting cloud resource demands and costs. Due to the many benefits of containers over Virtual Machines is the reason by 2022, more than 75% of organisations are expected to be running containerised applications within production (Gartner, 2020).

2.4.2 NIST Cybersecurity Framework Controls

The National Institute of Standards and Technology (NIST) is a government agency of the United States Department of Commerce. The NIST information lab is responsible for producing the gold standard of Cyber Security frameworks (Peacock, 2022).

Another essential NIST publication named SP 800-53 comprises over one thousand security controls to guide federal agencies to maintain and protect their IT security systems from potential security issues and cyber-attacks. The security controls from the SP 800-53 are categorised over twenty control families. These security control families function as foundational pillars for the NIST framework:

ID	Family	Num Controls	Relevant cloud, containers & K8S
AC	Access Control	147	16%
AU	Audit and Accountability	69	22%
AT	Awareness and Training	17	0%
CM	Configuration Management	66	27%
CP	Contingency Planning	56	0%
CA	Assessment, Authorization and Monitoring	32	22%
IA	Identification and Authentication	70	4%
IR	Incident Response	42	2%
MA	Maintenance	30	7%
MP	Media Protection	30	0%
PS	Personnel Security	18	0%
PE	Physical and Environmental Protection	59	0%
PL	Planning	17	0%
RA	Risk Assessment	26	12%
SA	System and Services Acquisition	145	25%
SI	System and Information Integrity	118	30%
SC	System and Communications Protection	162	20%
PM	Program Management	37	5%
PT	PII Processing and Transparency	21	0%
SR	Supply Chain Risk Management	27	0%

Table 1: NIST 800-53 (Rev. 5) families, number of controls, and percentage of controls relevant to cloud and container security (García, 2021).

From Table 1, it is clear that not all security control families from the NIST 800-53 publication are relevant to the security of containers. In this report, only relevant security controls will be analysed. The relevant security controls analysed in this paper are taken from the NIST SP 800-53 Revision 4 publication and are the most important for the security of container technologies (Souppaya, et al., 2017) .

2.4.2.1 Access controls (AC)

The Access Control (AC) Family comprises twenty-five security controls ranging from AC-1 to AC-25. However, within the AC control family, only five of the security controls are relevant to the security of container technology. The five security controls, AC-2, AC-3, AC-4, AC-6 and AC-17, are important to the security of containerised applications because they allow organisations to facilitate the implementation of access control policies. The usage of mandatory access control (MAC) policies increases the security of a containers pipeline. Without these policies in place, containers may be at high risk of exploitation. Implementing MAC policies can be a substantial mitigation practice against potentially system-wide compromises such as container runtime exploitation.

The Account Management Control (AC-2) aims to allow organisations to review and manage system accounts, including establishing, activating, modifying, disabling, and removing system accounts. The types of accounts managed by this control include individual, shared, group, system, guest, emergency, developer, temporary and service accounts.

Tools used to manage, scale and maintain containerised applications are called orchestrators. Orchestrators often include their own authentication directory service, which may differ from an organisation's specific directories that are currently in use. This may lead to weaker account management and potentially a system-wide compromise. Many of the accounts from the orchestrators are highly privileged and could potentially provide a significant risk for container technologies.

The Information Flow Enforcement (AC-4) control manages where data can travel within an organisation's information system. AC-4 achieves this by introducing restrictions to web requests, limiting unwanted traffic and regulating the transfer of data and information between organisations based on the organisation's own defined flow control policy. It is common practice for organisations to introduce an information flow control policy alongside enforcement methods to safely control the flow of information and data throughout their information systems. Enforcement methods include introducing hardware to allow one-way information flow and restricting data transfer between interconnected information systems.

The Least Privilege (AC-6) control ensures that the processes operate at privilege levels no higher than necessary to accomplish required organisational missions/business functions (VMware, 2021). Organisations employ the AC-6 control to be able only to authorise access to users and processes that are necessary, in accord with the current organisation's aims.

2.4.2.2 Audit and Accountability (AU)

As seen in Table 1, the Audit and Accountability security control family is one the most relevant control families to container security. The AU control family is in place to ensure that organisations introduce audit policies and procedures so that comprehensive assessment of company IT infrastructure can take place. This control family is important as the controls in place help assess company compliance to security measures within a production environment and help identify security vulnerabilities within a company's information system. Within the AU control family, there are sixteen security controls. The Audit Events (AU-2) control is an important policy that is put in place so that organisations can identify significant audit events, which can include password changes, failed logons, and admin privilege usage (VMWare, 2022).

Other security controls in the AU family that are vital for container technology security include AU-5, AU-6, AU-8, AU-9, and AU-12. Within the NIST Application Container Security Guide, to protect organisations against rogue containers, companies are encouraged to associate all container creation with individual user identities and should be logged (National Institute of Standards and Technology, 2017). Logging this information provides companies with a clear activity trail within their container's lifecycle.

2.4.2.3 Configuration management (CM)

Configuration management is another crucial security control family that introduces security policies that organisations should comply with to secure the process of adjusting default settings and mitigate risk. The Configuration Management control family consist of eleven security controls in total. The NIST 900-190 publication shows that the CM-2, CM-3, CM-4, CM-4, CM-5, CM-6, CM-7, and CM-9 controls are best used for container technology security (National Institute of Standards and Technology, 2017). A vital security control within the CM family is CM-3. The Configuration Change (CM-3) control covers configuration change and involves the systematic proposal, justification, implementation, testing review and system modifications (VMWare, 2022). Research from the NIST 800-190 publication noted that if containers are not put through the rigours of vulnerability scanning and proper configuration, they may be more susceptible to exploits (National Institute of Standards and Technology, 2017). This research stresses the importance of the Configuration Management security control family. Without proper configuration, images will be at a higher risk of containing exploits.

2.4.2.4 Assessment, authorisation, and monitoring (CA)

The CA control family consists of nine sub security controls and is in place within the NIST 800-53 publication to ensure and establish policies and procedures for implementing security controls. Recent research has shown that "The Assessment, authorisation and monitoring control family is implemented to ensure compliance with security policies and is critical to minimising the threat of breaches." (North Carolina Department of Information Technology, 2022). This research shows why companies should comply with the security policies provided within the CA family. In particular, the CA-9 control is vital to securing container technology security. The Internal System Connections (CA-9) security control is employed within container environments and is in charge of securing internal system connections. To mitigate risks against insufficient authentication and authorisation restrictions, organisations can configure their continuous integration process to allow images to be authorised and released to repositories only after they have passed vulnerability scanning and a compliance assessment (National Institute of Standards and Technology, 2017).

To protect organisations against image configuration defects, the Assessment, authorisation, and monitoring control policies also encourage organisations to implement continuous, updated monitoring of image compliance to detect any security weaknesses and risks at the organisational level. In doing so, organisations will be able to enforce compliance with configuration best practices leading to more reliable and secure software deployment. The NIST 800-190 publication also recommended that malware sets and behavioural detection heuristics be included within the monitoring process to defend containers and organisations against embedded malware (National Institute of Standards and Technology, 2017).

2.4.2.5 Identification and authentication (IA)

As seen in Table 1, the IA security control is only four per cent relevant to container security. However, IA-2, IA-4 and IA-5 controls are essential to uniquely identifying and authenticating organisational users and associating the identification with changes made to information systems. The identification policies use standard device identifiers like MAC and IP addresses to identify individual users, groups, roles or devices. These identification management policies are found within the IA-4 sub-control. On the other hand, the authentication management policies occur within the IA-5 control. The Identification and Authentication controls are essential to mitigating risks involving the use of untrusted images. Organisations are encouraged to use discrete identification of each image by cryptographic signatures using a validated NIST implementation (Souppaya, et al., 2017). The discrete identification of signatures can help organisations maintain a set of trusted images and help ensure secure images are run within environments to help mitigate the risks of malicious code being deployed into production environments.

2.4.2.6 Maintenance (MA)

Maintenance plays a vital part in protecting an organisation's information systems. Lack of compliance with NIST Maintenance controls can lead to outdated systems, increasing the risk of security breaches exponentially. Organisations must follow NIST Maintenance control procedures to prepare for potential security incidents within their containerised environments. Research from the NIST 800-190 suggests that organisations should implement ongoing monitoring and maintenance of container image repositories to ensure images are maintained and updated due to the regular changes to vulnerabilities and configuration requirements (National Institute of Standards and Technology, 2017). The ongoing maintenance of repositories can help mitigate the risks of untrusted image use within production environments.

2.4.2.7 System and services acquisition (SA)

Table 1 shows that the SA control family is one of the most relevant controls within the NIST 800-53 publication regarding the security of containers. The SA control family consists of 23 sub controls. The System and Services Acquisition control family is vital to container technology security. The policies aim to secure developer configuration management, security testing and evaluation. The Developer Configuration Management (SA-10) control requires developers to perform configuration management, document and manage the integrity of changes, and track security flaws within their system. Research from the NIST 800-53 publication states that organisations consider the quality and completeness of configuration management conducted by developers as direct evidence of applying effective security controls (NIST, 2020). This research suggests that effective security controls are directly correlated to the quality and completeness of a developer's configuration management. Developer Security Testing and Evaluation (SA-11) control is another important security control that is in place to secure developer testing and management. The SA-11 control is in place to ensure that required controls are implemented correctly, operating as intended, enforcing the desired security policies, and meeting established privacy requirements (NIST, 2020).

2.4.2.8 System and information integrity (SI)

The SI security control family consists of twenty-three sub controls that help develop, document and distribute organisation defined roles. The SI controls aim to help designate management roles for developing, documenting, and distributing information system integrity policies and procedures. The Flaw Remediation (SI-2) controls aim to identify, report and correct system flaws. Within containerised application development, flaw remediation plays an essential role in testing software and firmware updates and encouraging organisations to install newly tested security-relevant software and firmware updates.

The Information System Monitoring (SI-4) control is also helpful within a container's lifecycle. Its policies monitor systems for attacks, indicators, and unauthorised connections. The SI-4 control also aims to identify unauthorised system use and encourages organisations to invoke internal monitoring procedures to collect essential information. The Information System Monitoring (SI-4) control is relevant to container technology because organisations introducing monitoring devices, intrusion detection and prevention systems, malicious code protection software, and scanning tools within their pipeline can strengthen their system and data reliability.

As most images are sourced from public repositories, there is a risk of running poisoned images. The introduction of System and Information Integrity control family policies can help mitigate companies running poisoned images by validating the integrity of images before runtime by leveraging hashes and digital signatures (National Institute of Standards and Technology, 2017). There is a lower risk of deploying poisoned images into production environments with these mitigation strategies in place.

2.4.2.9 System and communications protection (SC)

The SC security control family consists of fifty-one sub controls that together help to protect organisational information systems and communications. The System and Communications Protection security controls aim to address role and responsibility coordination with policy, standards and regulation compliance. The SC security control family also aims to designate officials to manage the development, documentation and dissemination of the system and communication protection policies and procedures (NIST, 2020). According to the NIST Containerised Application Security Guide, SI-2, SI-4, and SI-7 are the most relevant controls for securing container technology within the SC security control family (National Institute of Standards and Technology, 2017). The implementation of system and communication protection can help mitigate container image risk. Numerous applications require secrets in order to enable secure communication. However, when apps are packaged within a container image, these secrets can be embedded directly into the image (National Institute of Standards and Technology, 2017). System and communication protection policies are in place so that organisations can protect against risks such as embedded secrets. Organisations that do not comply with SC controls can risk attackers parsing their images to learn the embedded secrets. The introduction of SC controls aims to provide security and protection to containerised application system communications by implementing boundary protection to monitor and control communications and preventing unauthorised information transfer via shared system resources.

2.4.2.10 Program Management (PM)

Program management security controls are implemented at the organisation level to manage an organisation's information security program. The PM security controls are in place to develop and deploy an organisation-wide information security program (NIST, 2020). The security program includes an overview of the security requirements and descriptions of the security management controls. Program Management security control also provides policies to review the security program plan and protect the plan from unauthorised disclosure and modification. The PM security control family consists of over thirty-two security controls to help organisations set up, review and protect their organisation-wide security program plan.

2.4.3 CI/CD Pipeline

A continuous integration/continuous delivery (CI/CD) pipeline is an automation process for software developers consisting of multiple steps that automate a software applications build, test and delivery process. "Automated pipelines can be a great way to secure the building and deployment of software." (National Cyber Security Centre, 2021). Within containerised application development, an organisation must take steps to secure the CI/CD pipeline to mitigate future security risks. DevSecOps practices can be utilised to secure the CI/CD pipeline and to build successful and secure software. Organisations need to secure their CI/CD pipeline as past research has shown that poor pipeline security configuration has led to a complete DevOps environment compromise (Haymore, 2022). A recent data breach has shown that misconfiguration can lead to substantial company data leaks. In early 2020, Estée Lauder's pipeline misconfiguration caused a massive database leak of upwards of 440 million personal records (SecureCloudDB, 2021). However, this research is not that surprising, as past research has also shown that misconfigurations are the most significant risk to cloud environments causing up to 70% of all security challenges in the cloud (Trend Micro, 2021). This shows how vital CI/CD pipeline security is and how just a tiny mistake within this process can lead to exponential data breaches.

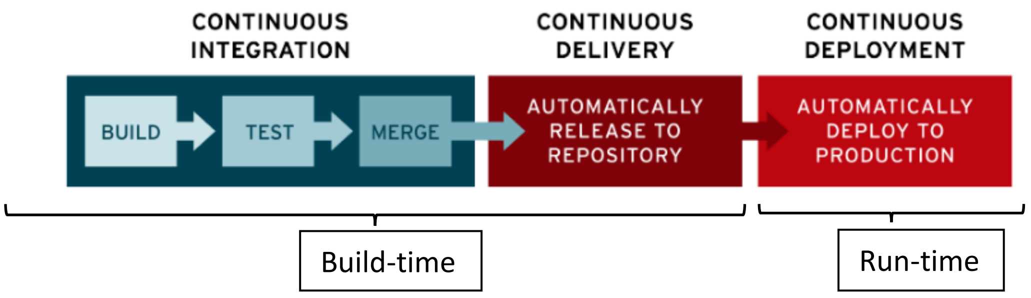


Figure 2, Basic CI/CD Pipeline Example (Red Hat, 2018)

Within a typical CI/CD pipeline, there is a series of stages: build, test, release, deployment and finally, the validation stage. Company pipelines may differ, but all follow the premise of these five stages. In the final validation stage, container image scanning tools scan images and reassure organisations by comparing the image operating systems and libraries to known CVEs. Research has shown that "an insecure CI/CD pipeline will inevitably lead to an insecure application." (Sysdig, 2022). Organisations should take steps to manage security risks during all stages of their CI/CD pipeline by introducing security controls, build-time security and runtime security to minimise the risk of insecure software.

2.4.3.1 Build-time Security & Access Controls

Within the building stage of software, companies must follow security policies and controls to secure the software and monitor the process of pushing the software through the pipeline. Build-time security ensures that company software is safely and securely pushed to production environments. Controls from security frameworks, such as the NIST Framework, are introduced within the build stage to secure software production within the pipeline. Controls such as Information Flow Enforcement (AC-4) and Least Privilege (AC-6) from the Access Control family are implemented for build-time security as staff within organisations must be handed privileges levels that are no higher than necessary. Account management controls must be implemented within the build stage to monitor and review system account usage.

2.4.3.1.1 Build stage

Account Management (AC-2) control policies should be implemented within the build stage of the pipeline to ensure that system accounts are managed securely. This will include dynamic privilege management, automated audit actions, and shared and group account restrictions. The NIST 800-53 recommends that organisations should prohibit the use of shared, group, temporary and guest accounts due to their increased risk to security (NIST, 2020).

Audit and Accountability controls should also be implemented throughout the build stage in order to provide organisations with audit logs of significant events such as password changes and admin privilege use. AU controls within the build stage are to help organisations assess compliance with security policies and identify any vulnerabilities within their information systems. Identification and Authentication

(IA) should be used in conjunction with Audit and Accountability (AU) controls to identify and authenticate changes made in container build stages. With the implementation of Access Control (AC), Audit and Accountability (AU) and Identification and Authentication (IA) security controls, the build stage can be secured to help mitigate security risks.

2.4.3.1.2 Test stage

System and Service Acquisition (SA) control policies should be implemented within the test stage of the pipeline to ensure secure developer configuration management and testing. System and Services Acquisition (SA) controls such as SA-10 and SA-11 should be introduced within the test stage to ensure that developer testing and monitoring are done securely.

System and Information Integrity (SI) control policies should also be introduced within the testing stage of the pipeline. The role of SI controls within the test stage is to strengthen their system and data reliability by introducing monitoring tools and malicious code detection.

2.4.3.1.3 Merge stage

Application security tools such as Jfrog Xray and Prisma Cloud provide Software Composition Analysis (SCA) that introduces security policies that prevent images from being deployed into production environments if the image contains a certain CVE severity level. Organisations should use application security SCA tools to increase build-time security. Preventing vulnerable software from being pushed to repositories is vital; as stated before, past research has shown that over half of the images within the Docker Hub Repository contain a critical CVE (Barua, 2020). Increasing the use of SCA tools within company pipelines will help combat vulnerable images being pushed to public repositories and help prevent potential catastrophic data breaches. Vulnerable images must not be pushed to repositories as an organisation may unknowingly run a vulnerable image under the pretence that it is secure. Without further image analysis and testing, deploying these vulnerable images may cause system-wide compromises in the future.

2.4.3.2 Runtime security

Runtime security is the act of preventing exploits within live application environments. Runtime security takes place in the final stage of the CI/CD pipeline. When deploying an image into a production environment, it is crucial to run live tests on the image, as static image analysis does not provide a complete insight into all possible exploits within an image. Some exploits only show when an image is in a live environment. Further security precautions should be taken to run further tests, such as scanning live images using dynamic vulnerability assessment tools. Organisations must run regular tests on active images to increase runtime security and keep up to date with zero-day exploits. A zero-day exploit is a brand new unknown vulnerability that has only been exposed for less than a day. As time goes on, more and more vulnerabilities and exploits within package libraries and operating systems are uncovered. Regular tests must be run to combat the constant pressures of zero-day exploits.

2.4.4 Role of Vulnerability Assessment Tools

The use of vulnerability assessment tools within the lifecycle of containerised applications is crucial. Past research has shown that vulnerability assessment tools should be used throughout all stages of a container's lifecycle to help prevent any potential attacks and stop attackers from tampering with company deployments (Rice, 2021). Organisations use vulnerability assessment tools as a last precaution before pushing images to their production environments. One of the last steps in a CI/CD pipeline is to push images through vulnerability assessment tools to find any known CVEs. Organisations must run both static and active image tests as some exploits only appear when an image has been run and are not findable when the image is static. Most tools are specific to running tests on static or active images; however, a select few have been found to run both. Recent research into the area of Docker Container Security within Cloud Computing has provided exploration into the effectiveness of dynamic analysis assessment tools against vulnerable container images. However, there is a lack of research on static image analysis (Brady, et al., 2020). Therefore, this paper aims to run static container image tests on active and depreciated images, to understand the effectiveness of static vulnerability assessment tools.

The role of the vulnerability assessment tools is to uncover known vulnerabilities and exploits within container images. The tools scan operating system and non-operating system libraries for vulnerabilities. The tools compare the libraries to a regularly updated database filled with CVEs. The CVEs found within container images are then ranked using a severity level Common Vulnerability Scoring System (CVSS). This CVSS system is used as an industry standard to provide vulnerabilities with an access severity level. The level given to a discovered exploit is ranked out of ten and then listed from Low-Critical depending on the CVSS access complexity given to the exploit. The access complexity measures how complex the attack is required to exploit the vulnerability found. Most organisations commonly configure their vulnerability assessment tools to discover high severity level CVEs as these pose the most threat to their information systems and pipelines. However, research has shown that CVEs with a low complexity level are a growing concern and should not be disregarded (Scroxton, 2021). This research has shown that the growing concern for low complexity CVEs is due to their proneness to mass exploitation and gives the more highly skilled attackers time to save their high complexity zero-day exploits for the future.

CVE Rating	CVSS Score
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Table 2, CVSS v3.0 Ratings

3 Design, Investigation and Results

The project aims to provide greater insight into container security controls by analysing the reliability and effectiveness of popular open-source container vulnerability assessment tools. This paper will run multiple images through popular open-source vulnerability assessment tools to compare the number of vulnerabilities found per tool. An in-depth analysis will be done on each image's low, medium, high, and critical risks. The analysis will use the Common Vulnerability Scoring System (CVSS) and the Common Vulnerabilities and Exposures (CVE) list to distinguish which of the three tools is the most effective at finding vulnerabilities within Docker Images. A Linux Ubuntu OS will be set up on a Virtual Machine (VM) to install the tools and run the tests.

3.1 Vulnerability Assessment Tools

Three popular open-source vulnerability assessment tools were chosen to test the Docker images during the investigation.

3.1.1 Grype

The first open-source vulnerability assessment tool is from a software security company named Anchore. Grype is a vulnerability scanner for both container images and filesystems. The tool works alongside a command-line interface (CLI) tool named Syft that uses a library to generate a Software Bill of Materials (SBOM) to detect vulnerabilities within containers. An SBOM provides an insight into a list of components for a piece of software.

When testing a container image through Grype, the tool's first step is to check for any updates in the vulnerability database to ensure that any newly found vulnerabilities have been added. After updating the vulnerability database, the next step is to pull the image from the Docker repository. Once the image has been fully extracted, the image is then loaded into the tool and analysed. The software components are extracted and scanned individually for any CVEs from the vulnerability database following the image analysis.

Once the image has gone through the scanning process, each software component with known vulnerabilities is listed in a table. The table shows the name of the software component at risk, the version of the software component that poses a risk, the CVE vulnerability ID, and the severity of the vulnerability found. The table will also show if the vulnerability has been fixed in an updated version of the software component.

3.1.2 Trivy

The second tool is from the cloud-native security company named Aqua Security. Trivy is an open-source vulnerability assessment tool that scans for vulnerabilities in container images, file systems, and Github repositories and can also scan for issues within configurations (Github, 2022). On top of scanning for vulnerabilities within OS packages and language packages, Trivy also detects potential configuration issues within Infrastructure as Code (IaC) files to be able to mitigate any possible risks to deployments.

Like Grype, the tool automatically downloads and updates the vulnerability database before beginning the vulnerability detection once the scanning process begins. Once the database is up to date, the tool then detects the base OS of the image. Following this, Trivy then begins detecting OS-specific vulnerabilities within the image. After analysing the OS packages, Trivy identifies the language-specific files such as Java and Yarn within the image and detects vulnerabilities specific to the language packages found within the image.

Once an image has been scanned, Trivy provides a bordered table of results. The table shows the title of each library, the CVE ID, the severity of the vulnerability found, the installed version of the library, the version of the library in which the vulnerability is fixed in, and a brief description of what has gone wrong within the library to cause the exploit. Trivy also makes it easy to find out the total number of vulnerabilities found by including a total of low, medium, high, and critical vulnerabilities found in the scan.

3.1.3 Dagda

Dagda is the last open-source vulnerability assessment tool used in the paper. Dagda provides static analysis on container images for known vulnerabilities within its unique database. The tool requires the user to manually create and populate the database using the software MongoDB. The advantage Dagda has over the other two vulnerability tools used in this paper is that it scans for CVEs, Bugtraq IDs, Red Hat Security Advisories (RHSA) and Red Hat Bug Advisories (RHBA).

Image analysis can begin once the mongo database has been set up and populated with the latest known vulnerabilities. Dagda works similarly to Trivy, pulling information from OS packages, programming language dependencies, and verifying software versions. Dagda detects vulnerabilities within the container, but the tool also uses the ClamAV antivirus engine to detect trojans, viruses, and malware that may be included within containerised environments. Dagda's extra steps to provide top-level container security may be why recent research shows Dagda may be the best solution for static analysis due to its small resource demand and the ability to analyse running containers (Czikó, 2019).

3.2 VULNERABILITY ASSESSMENT TOOL SETUP

3.2.1 Container Image Collection

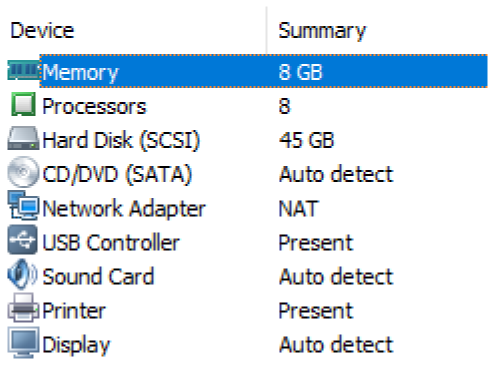
Twenty container images from within the Docker Hub Repository were selected from multiple categories varying from operating systems, messaging services and programming languages. The same twenty Docker images were used in testing across all tools to provide fair results. Four out of the twenty images were specifically chosen to be discontinued versions as they should provide an insight into the risks of using outdated images in an organisation's pipeline.

3.2.2 Virtual Machine Setup

When deciding on which OS to run the tests on, it was essential to ensure the tools and their software dependencies could be installed together. After researching the three tools used within this paper, it was decided that Linux Ubuntu would be used as the OS.

Setting up the Virtual Machine was straightforward and did not require much configuration. In this paper, Linux Ubuntu version 18.04.5 LTS was used as all the tools can run on this version.

A Linux Ubuntu image was downloaded from the official Linux website. When installing and creating the VM, it was essential to ensure that it had more CPU cores and allocated RAM than the default allocation on a standard install. A decision was made to allocate 8GB of RAM and 8 CPU cores to the VM to ensure that the tools had enough memory and cores to run smoothly and effectively. Without increasing the RAM and CPU allocation, the tools will not be able to run correctly and sometime not at all.



Device	Summary
Memory	8 GB
Processors	8
Hard Disk (SCSI)	45 GB
CD/DVD (SATA)	Auto detect
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Figure 3, Virtual Machine Hardware Setup

After setting up the hardware, a few commands were run to make sure the base software on the VM was up to date. The commands include: 'sudo apt update' and 'sudo apt upgrade'.

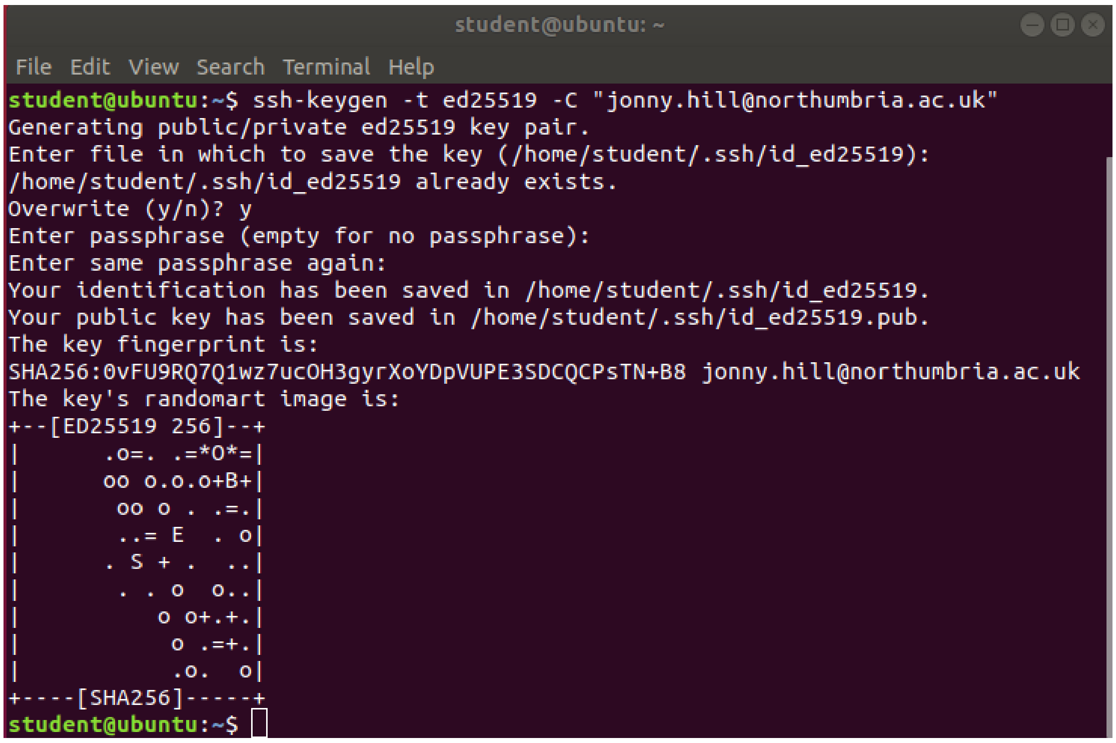
The update command is used to download the up to date package lists from the repositories needed for upgrading the software and updates them to the newest versions. Following the update command, the upgrade command is used to fetch the newly updated packages and install any updates. This is done to ensure that the VMs software is up to date with the latest versions. Keeping the software up to date is crucial because the new updates to the software improve their performance and fix any known bugs that could cause problems later in the vulnerability tool setup.

3.2.3 Basic Software Dependencies

After collecting the container images, the next step was to set up the software dependencies. This was an essential step within the tool setup as, without the following software, the tools would not be able to be installed on the Linux VM.

3.2.3.1 Github SSH Key Setup

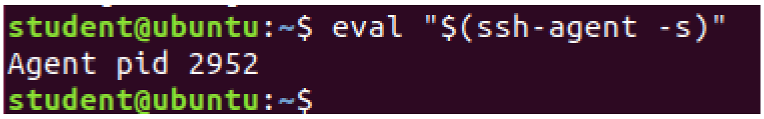
When installing the tools used in this project, they are pulled from their Github repositories. As a result, an SSH connection must be made between the Linux VM and Github. A new SSH key must be generated through the Linux terminal to begin setting up the SSH connection. A new SSH key has been generated using the provided email as a label in the figure below.



```
student@ubuntu:~$ ssh-keygen -t ed25519 -C "jonny.hill@northumbria.ac.uk"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/student/.ssh/id_ed25519):
/home/student/.ssh/id_ed25519 already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/student/.ssh/id_ed25519.
Your public key has been saved in /home/student/.ssh/id_ed25519.pub.
The key fingerprint is:
SHA256:0vFU9RQ7Q1wz7ucOH3gyrXoYDpVUPE3SDCQCPsTN+B8 jonny.hill@northumbria.ac.uk
The key's randomart image is:
+--[ED25519 256]--+
|      .o=. .=*O*=|
|      oo o.o.o+B+|
|       oo o . .=.|
|       ..= E  . o|
|      . S + .  ..|
|       . . o  o..|
|          o o+.+.|
|           o .=+.|
|           .o.  o|
+----[SHA256]-----+
student@ubuntu:~$
```

Figure 4, SSH Key Generation

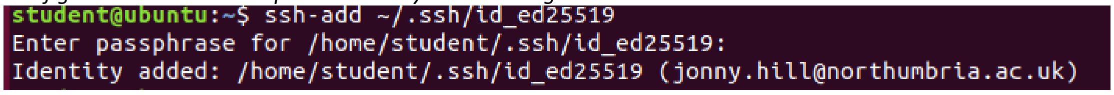
Once an SSH key has been generated for the Linux machine, the key must be added to the ssh-agent. Firstly, the ssh-agent must be running in the background before the SSH key can be added. The figure below shows the command run to start the ssh-agent.



```
student@ubuntu:~$ eval "$(ssh-agent -s)"
Agent pid 2952
student@ubuntu:~$
```

Figure 5, Starting the ssh-agent

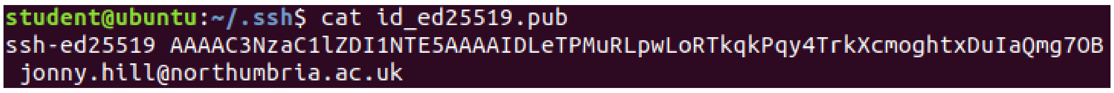
Once the ssh-agent is running, the private SSH key that we generated previously can be added to the ssh-agent. The figure below shows the private SSH key to the ssh-agent.



```
student@ubuntu:~$ ssh-add ~/.ssh/id_ed25519
Enter passphrase for /home/student/.ssh/id_ed25519:
Identity added: /home/student/.ssh/id_ed25519 (jonny.hill@northumbria.ac.uk)
```

Figure 6, Adding the private SSH key to the ssh-agent

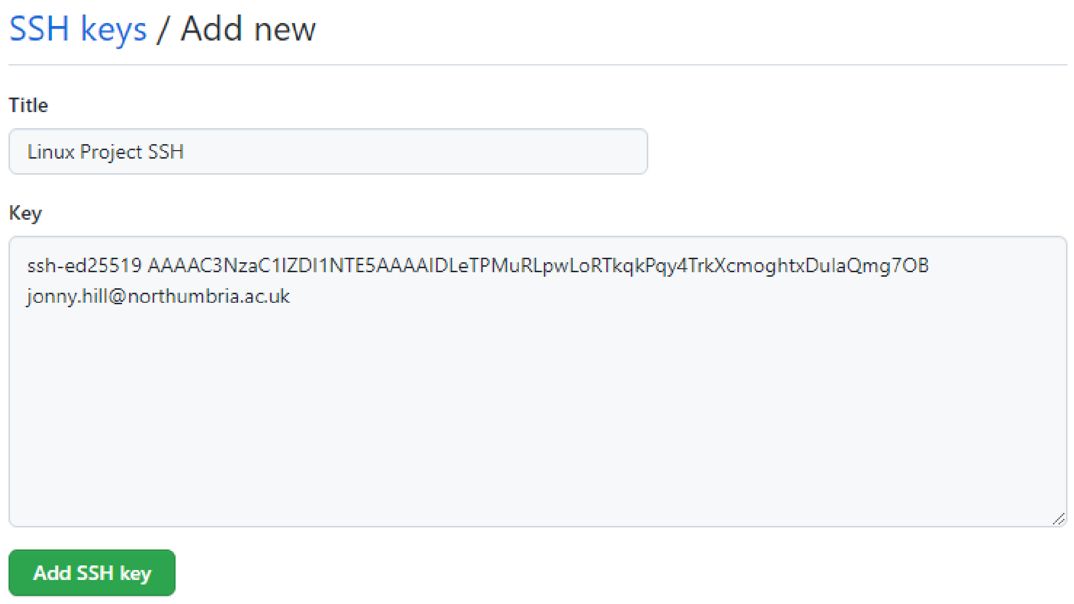
After the private key has been added, a connection can be made by uploading the public key that has been generated from the ssh-agent. The figure below shows how the public key was found.



```
student@ubuntu:~/.ssh$ cat id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDLeTPMuRLpwLoRTkqkPqy4TrkXcmoghtxDuIaQmg7OB
 jonny.hill@northumbria.ac.uk
```

Figure 7, Finding newly generated public SSH key

The new public SSH key is then added to GitHub within the SSH section settings of the Github account. The figure below shows the public key being added to the GitHub account.



SSH keys / Add new

Title

Linux Project SSH

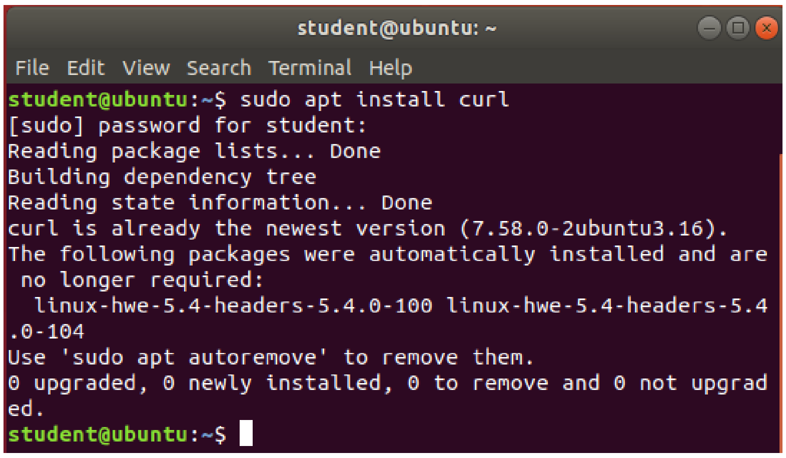
Key

ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDLeTPMuRLpwLoRTkqkPqy4TrkXcmoghtxDuIaQmg7OB
jonny.hill@northumbria.ac.uk

Figure 8, Adding public SSH key to Github

3.2.3.2 Curl Install

Curl is a software that is used within the command line terminal of Linux to provide easy and quick installation of software by transferring data through a variety of network protocols. The figure below shows the command used to install Curl.

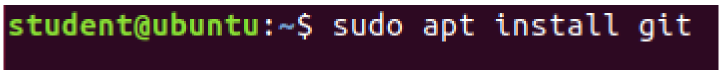


```
student@ubuntu:~$ sudo apt install curl
[sudo] password for student:
Reading package lists... Done
Building dependency tree
Reading state information... Done
curl is already the newest version (7.58.0-2ubuntu3.16).
The following packages were automatically installed and are
 no longer required:
  linux-hwe-5.4-headers-5.4.0-100 linux-hwe-5.4-headers-5.4
.0-104
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgrad
ed.
student@ubuntu:~$
```

Figure 9, Curl Installation

3.2.3.3 Git Install

For tools that did not provide a simple curl install command, a Github repository clone was required instead. In order to clone the repo, the software Git was required. The figure below shows the command used to install Git.



```
student@ubuntu:~$ sudo apt install git
```

Figure 10, Git Installation

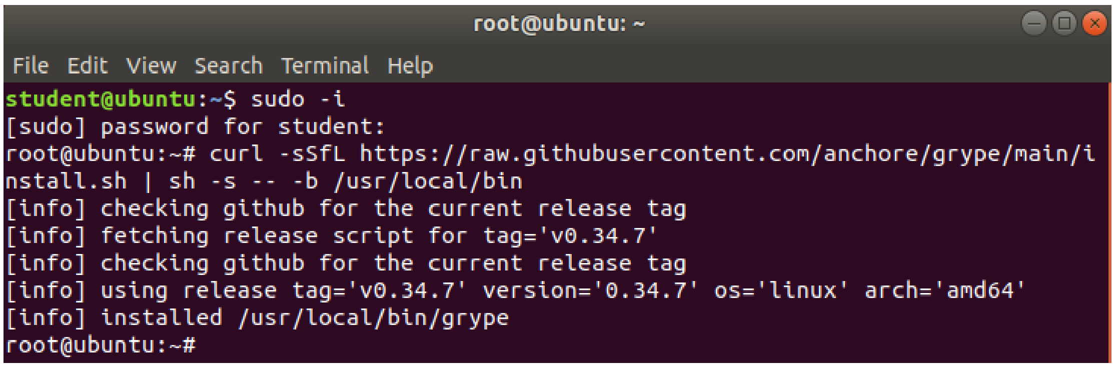
3.2.4 Tool Setup

After setting up the software dependencies, the next step was installing each vulnerability assessment tool. Once the vulnerbality assessment tools have been installed the container image tested can start.

3.2.4.1 Grype

When installing Grype, the SSH authentication needed to be set up correctly to be able to pull the tool from the Github repositories. To begin the installation of the Grype, a terminal was opened, and the command 'sudo -i' was run so that the terminal had root user privileges. This was important as the software was installed within the 'usr/local/bin' directory. This directory is used for programs that any user on the VM will be able to run and is the default install location from the Grype install guide.

Once the terminal had root privileges, a curl command was run to begin the installation.

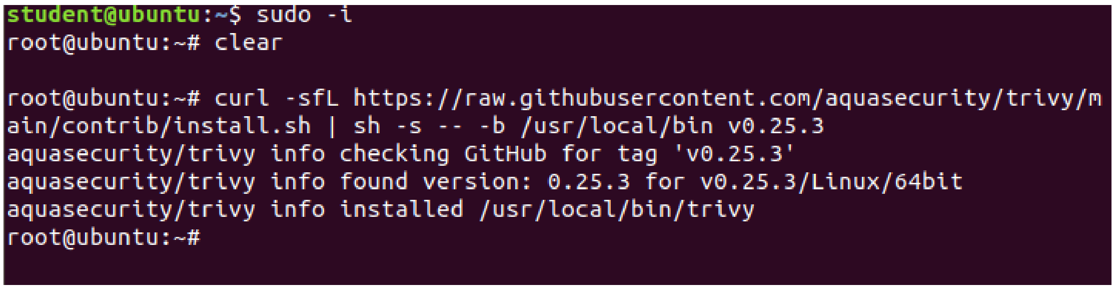


```
student@ubuntu:~$ sudo -i
[sudo] password for student:
root@ubuntu:~# curl -sSfL https://raw.githubusercontent.com/anchore/grype/main/i
nstall.sh | sh -s -- -b /usr/local/bin
[info] checking github for the current release tag
[info] fetching release script for tag='v0.34.7'
[info] checking github for the current release tag
[info] using release tag='v0.34.7' version='0.34.7' os='linux' arch='amd64'
[info] installed /usr/local/bin/grype
root@ubuntu:~#
```

Figure 11, Grype Installation

3.2.4.2 Trivy

The installation of the Trivy tool was a similar process to the Grype installation. Once the SSH connection between the VM and Github was made, and Curl was installed, the installation of Trivy could take place with a single command in the root terminal.



```
student@ubuntu:~$ sudo -i
root@ubuntu:~# clear

root@ubuntu:~# curl -sfL https://raw.githubusercontent.com/aquasecurity/trivy/m
ain/contrib/install.sh | sh -s -- -b /usr/local/bin v0.25.3
aquasecurity/trivy info checking GitHub for tag 'v0.25.3'
aquasecurity/trivy info found version: 0.25.3 for v0.25.3/Linux/64bit
aquasecurity/trivy info installed /usr/local/bin/trivy
root@ubuntu:~#
```

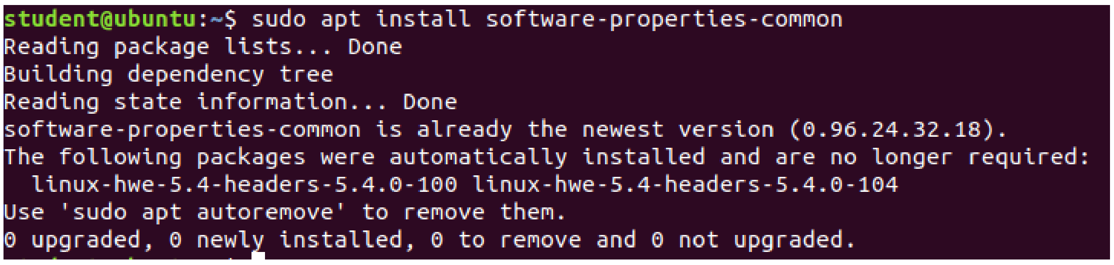
3.2.4.3 Dagda

3.2.4.3.1 Software Requirements

Before installing the Dagda tool, additional software dependencies are required for the tool to work.

3.2.4.3.1.1 Python3.8

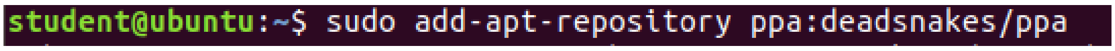
Before installing python3.8, further software prerequisites are required before Python can be installed. A software named 'software-properties-common' is required. This software provides scripts that make installing Personal Package Archives (PPA) easier. The figure below shows the command run to install the python3.8 prerequisites.



```
student@ubuntu:~$ sudo apt install software-properties-common
Reading package lists... Done
Building dependency tree
Reading state information... Done
software-properties-common is already the newest version (0.96.24.32.18).
The following packages were automatically installed and are no longer required:
  linux-hwe-5.4-headers-5.4.0-100 linux-hwe-5.4-headers-5.4.0-104
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

Figure 12, software-properties-common install

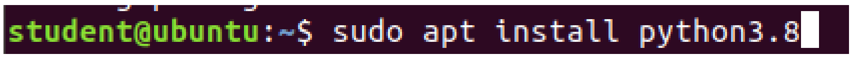
After, a PPA is installed to allow the Linux machine to run multiple versions of Python. The figure below shows the installation of the deadsnakes PPA.



```
student@ubuntu:~$ sudo add-apt-repository ppa:deadsnakes/ppa
```

Figure 13, Deadsnakes PPA installation

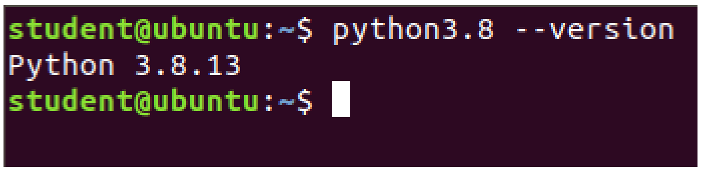
After installing the PPA, Python3.8 can be installed on the machine. The figure below shows the installation of Python3.8.



```
student@ubuntu:~$ sudo apt install python3.8
```

Figure 14, Python3.8 installation

After installing Python, a simple version check is done to ensure that Python3.8 is installed on the system. The figure below shows the command run to check if the version of Python was installed correctly.

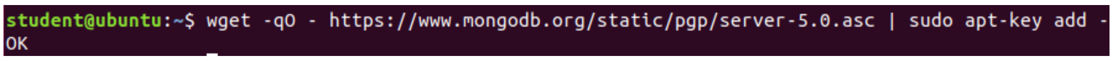


```
student@ubuntu:~$ python3.8 --version
Python 3.8.13
student@ubuntu:~$
```

Figure 15, Checking Python3.8 Install

3.2.4.3.1.2 MongoDB

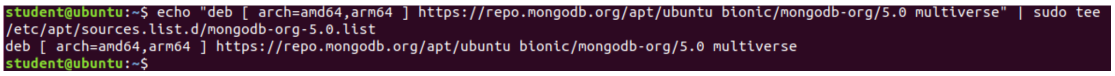
The second software required by the Dagda Tool was MongoDB. This piece of software was required to create the vulnerability database. Firstly, a public key was imported. The package management system uses this public key. This is shown in the figure below.



```
student@ubuntu:~$ wget -qO - https://www.mongodb.org/static/pgp/server-5.0.asc | sudo apt-key add -
OK
```

Figure 16, Public Key Import Command

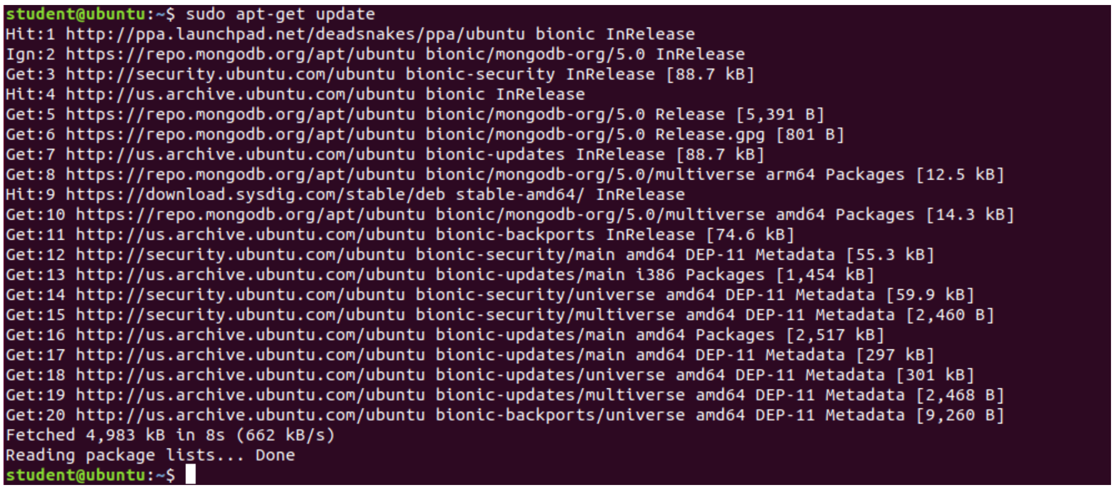
After the public key is imported, a list is required for the Mongo Database. The list is created with the command shown in the figure below.



```
student@ubuntu:~$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/5.0 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-5.0.list
deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/5.0 multiverse
student@ubuntu:~$
```

Figure 17, MongoDB List Creation Command.

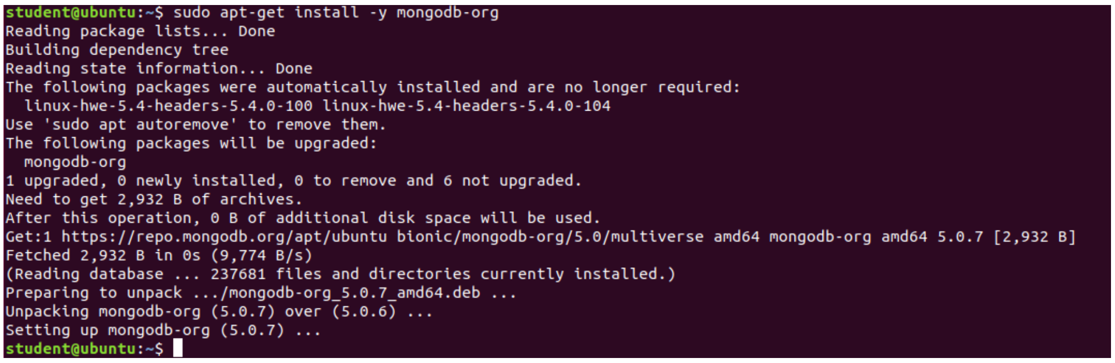
Next, the 'sudo apt-get update' command is run to reload the local package database.



```
student@ubuntu:~$ sudo apt-get update
Hit:1 http://ppa.launchpad.net/deadsnakes/ppa/ubuntu bionic InRelease
Ign:2 https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/5.0 InRelease
Get:3 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:4 http://us.archive.ubuntu.com/ubuntu bionic InRelease
Get:5 https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/5.0 Release [5,391 B]
Get:6 https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/5.0 Release.gpg [801 B]
Get:7 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:8 https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/5.0/multiverse arm64 Packages [12.5 kB]
Hit:9 https://download.sysdig.com/stable/deb stable-amd64/ InRelease
Get:10 https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/5.0/multiverse amd64 Packages [14.3 kB]
Get:11 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:12 http://security.ubuntu.com/ubuntu bionic-security/main amd64 DEP-11 Metadata [55.3 kB]
Get:13 http://us.archive.ubuntu.com/ubuntu bionic-updates/main i386 Packages [1,454 kB]
Get:14 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 DEP-11 Metadata [59.9 kB]
Get:15 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 DEP-11 Metadata [2,460 B]
Get:16 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [2,517 kB]
Get:17 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 DEP-11 Metadata [297 kB]
Get:18 http://us.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 DEP-11 Metadata [301 kB]
Get:19 http://us.archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 DEP-11 Metadata [2,468 B]
Get:20 http://us.archive.ubuntu.com/ubuntu bionic-backports/universe amd64 DEP-11 Metadata [9,260 B]
Fetched 4,983 kB in 8s (662 kB/s)
Reading package lists... Done
student@ubuntu:~$
```

Figure 18, Reloading the local package database

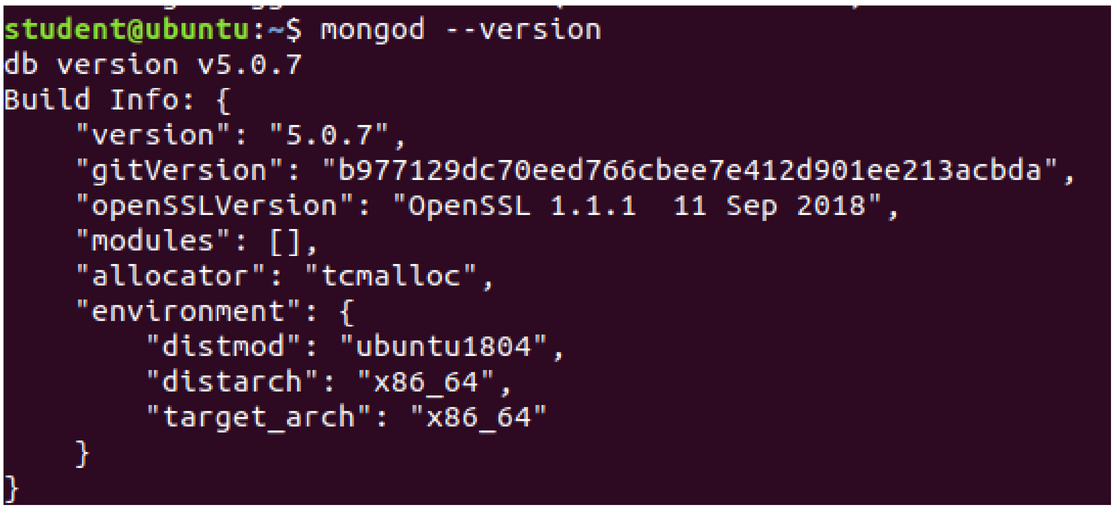
The next step is to install the MongoDB packages. This is shown in the figure below.



```
student@ubuntu:~$ sudo apt-get install -y mongodb-org
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-hwe-5.4-headers-5.4.0-100 linux-hwe-5.4-headers-5.4.0-104
Use 'sudo apt autoremove' to remove them.
The following packages will be upgraded:
  mongodb-org
1 upgraded, 0 newly installed, 0 to remove and 6 not upgraded.
Need to get 2,932 B of archives.
After this operation, 0 B of additional disk space will be used.
Get:1 https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/5.0/multiverse amd64 mongodb-org amd64 5.0.7 [2,932 B]
Fetched 2,932 B in 0s (9,774 B/s)
(Reading database ... 237681 files and directories currently installed.)
Preparing to unpack .../mongodb-org_5.0.7_amd64.deb ...
Unpacking mongodb-org (5.0.7) over (5.0.6) ...
Setting up mongodb-org (5.0.7) ...
student@ubuntu:~$
```

Figure 19, Installing MongoDB packages

Finally, a simple version check can be run once the packages have been installed to ensure that the installation has been completed.

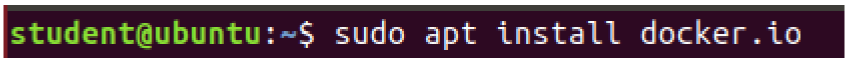


```
student@ubuntu:~$ mongod --version
db version v5.0.7
Build Info: {
    "version": "5.0.7",
    "gitVersion": "b977129dc70eed766cbee7e412d901ee213acbda",
    "openSSLVersion": "OpenSSL 1.1.1  11 Sep 2018",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "ubuntu1804",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}
```

Figure 20, MongoDB Version

3.2.4.3.1.3 Docker

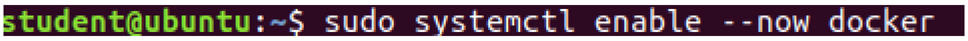
The figure below shows the command run to install the Docker engine.



```
student@ubuntu:~$ sudo apt install docker.io
```

Figure 21, Docker Engine installation

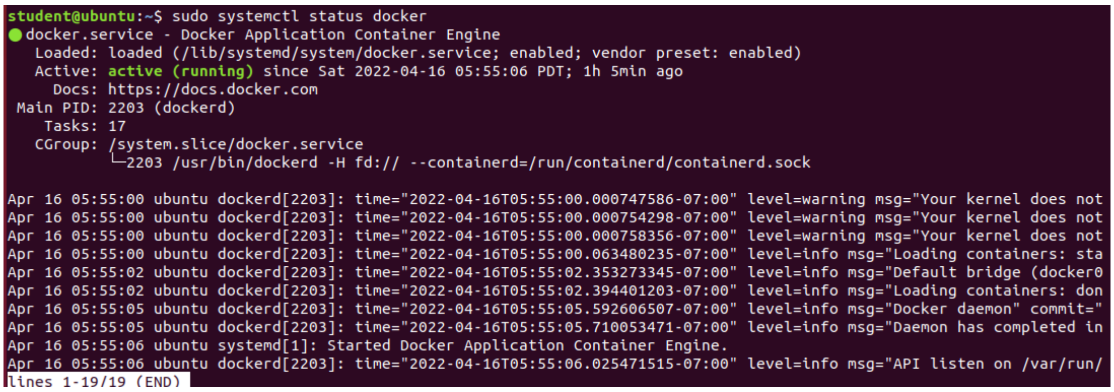
After installing the Docker, a command is run to start the Docker engine and allow the Docker Engine service to start on boot. The figure below shows the command to enable the Docker service to start on system boot.



```
student@ubuntu:~$ sudo systemctl enable --now docker
```

Figure 22, Enabling Docker Service on boot

After enabling the Docker service, a check is done to ensure that the Docker Engine is actively running in the background. This check is done to ensure that no errors have taken place in the installation. The figure below shows the command run to check the service.

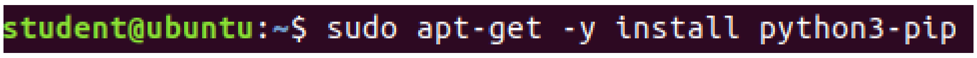


```
student@ubuntu:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2022-04-16 05:55:06 PDT; 1h 5min ago
     Docs: https://docs.docker.com
 Main PID: 2203 (dockerd)
    Tasks: 17
   CGroup: /system.slice/docker.service
           └─2203 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Apr 16 05:55:00 ubuntu dockerd[2203]: time="2022-04-16T05:55:00.000747586-07:00" level=warning msg="Your kernel does not
Apr 16 05:55:00 ubuntu dockerd[2203]: time="2022-04-16T05:55:00.000754298-07:00" level=warning msg="Your kernel does not
Apr 16 05:55:00 ubuntu dockerd[2203]: time="2022-04-16T05:55:00.000758356-07:00" level=warning msg="Your kernel does not
Apr 16 05:55:00 ubuntu dockerd[2203]: time="2022-04-16T05:55:00.063480235-07:00" level=info msg="Loading containers: sta
Apr 16 05:55:02 ubuntu dockerd[2203]: time="2022-04-16T05:55:02.353273345-07:00" level=info msg="Default bridge (docker0
Apr 16 05:55:02 ubuntu dockerd[2203]: time="2022-04-16T05:55:02.394401203-07:00" level=info msg="Loading containers: don
Apr 16 05:55:05 ubuntu dockerd[2203]: time="2022-04-16T05:55:05.592606507-07:00" level=info msg="Docker daemon" commit="
Apr 16 05:55:05 ubuntu dockerd[2203]: time="2022-04-16T05:55:05.710053471-07:00" level=info msg="Daemon has completed in
Apr 16 05:55:06 ubuntu systemd[1]: Started Docker Application Container Engine.
Apr 16 05:55:06 ubuntu dockerd[2203]: time="2022-04-16T05:55:06.025471515-07:00" level=info msg="API listen on /var/run/
lines 1-19/19 (END)
```

Figure 23, Checking the Docker service status

3.2.4.3.1.4 Pip3

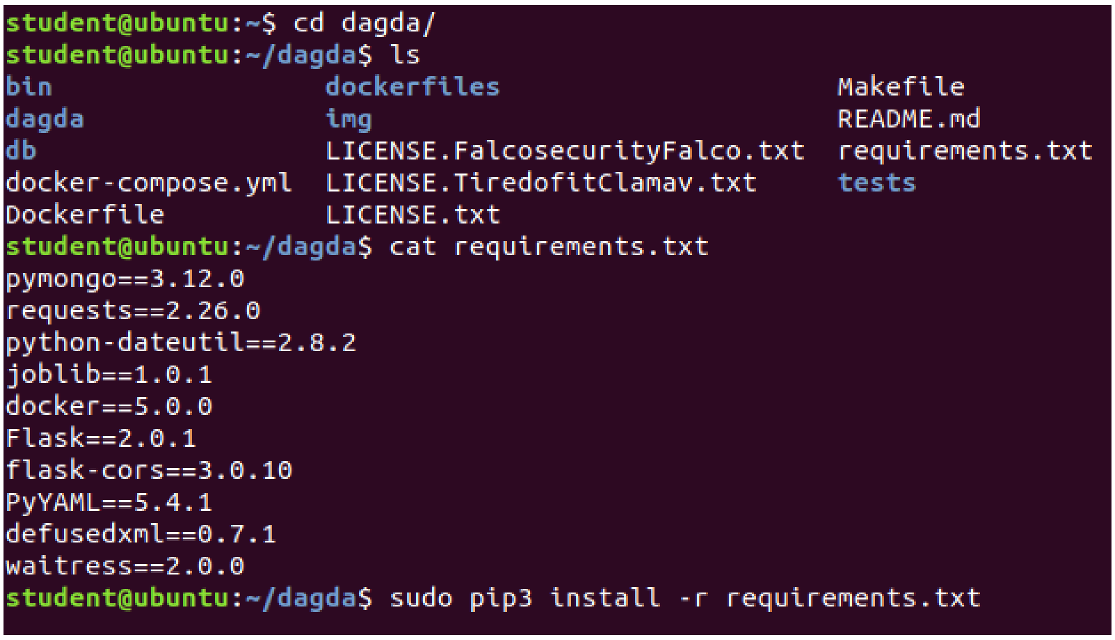
The figure below shows the installation command for the pip3 software.



```
student@ubuntu:~$ sudo apt-get -y install python3-pip
```

Figure 24, pip3 installation

After installing Pip3, more software requirements were required and could be easily installed within one command. Once the other software dependencies were installed, along with the installation of the Dagda tool, within the Dagda directory, a text file contained all the final software requirements. A terminal can be opened within this directory, and a pip command can be run to install all the final requirements for the tool. The figure below shows the process of installing the final software dependencies.

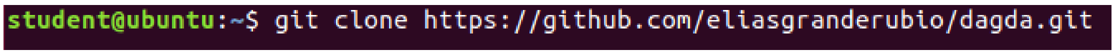


```
student@ubuntu:~$ cd dagda/
student@ubuntu:~/dagda$ ls
bin                   dockerfiles                      Makefile
dagda                 img                              README.md
db                    LICENSE.FalcosecurityFalco.txt   requirements.txt
docker-compose.yml    LICENSE.TiredofitClamav.txt      tests
Dockerfile            LICENSE.txt
student@ubuntu:~/dagda$ cat requirements.txt
pymongo==3.12.0
requests==2.26.0
python-dateutil==2.8.2
joblib==1.0.1
docker==5.0.0
Flask==2.0.1
flask-cors==3.0.10
PyYAML==5.4.1
defusedxml==0.7.1
waitress==2.0.0
student@ubuntu:~/dagda$ sudo pip3 install -r requirements.txt
```

Figure 25, Installing software from the requirements text file

3.2.4.3.2 Tool Configuration

A git clone command is used to copy the Github repository to the Dagda directory on the Linux machine to install Dagda. The figure below shows the command used to install the Dagda tool.



```
student@ubuntu:~$ git clone https://github.com/eliasgranderubio/dagda.git
```

Figure 26, Installing Dagda

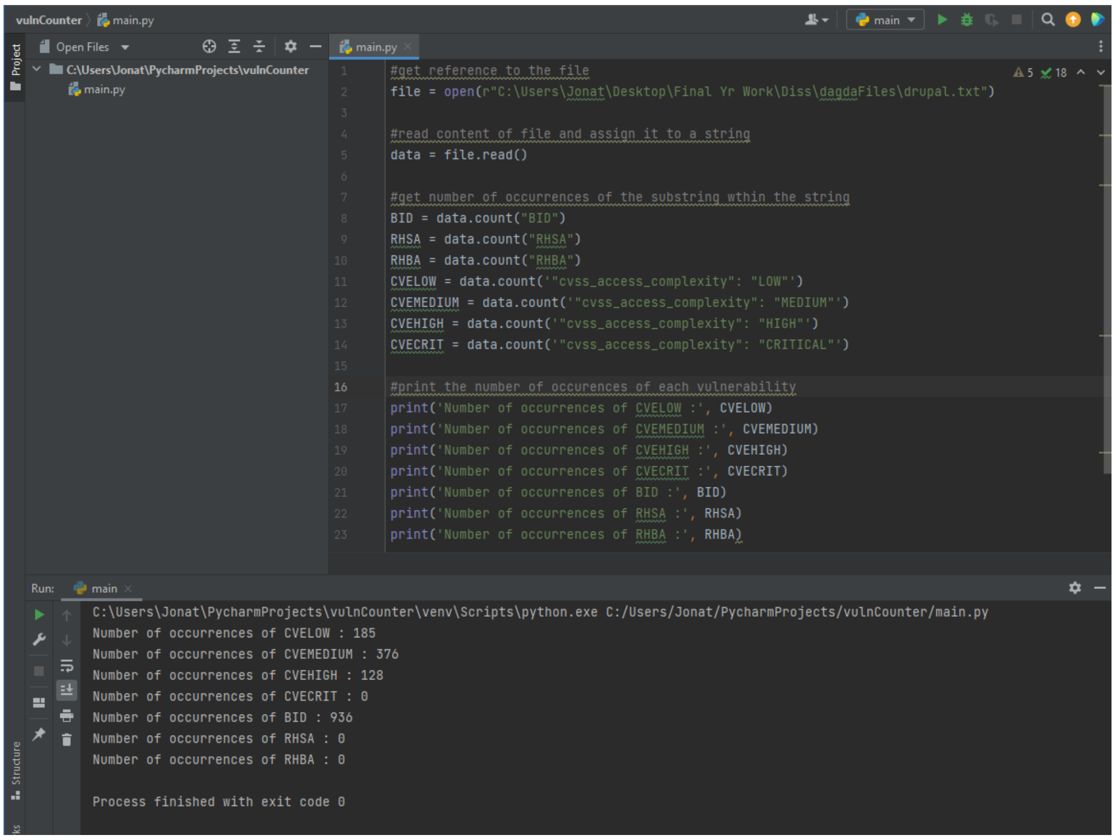
3.2.5 Design Challenges

This section will provide the challenges faced within the practical investigation.

3.2.5.1 Counting up total vulnerabilities for Dagda

When testing an image with the Dagda tool, the results were returned within the CLI. This posed a problem because many images returned over a thousand different vulnerabilities, making it difficult to count each exploit. To overcome this design challenge, a simple python program was created. Firstly, the output for each container image test was redirected to an individual text file. The program worked by opening a text file containing the returned result of a container image test. The program then counted the number of occurrences of each possible exploit found using the Dagda Tool.

This text file is then set as a variable named 'file'. This variable is then read and set to a string under a different variable name, 'data'. Following this, multiple lines of code were created for each type of exploit that the Dagda tool could find. Each line counted the data variable to get the number of occurrences of the substring within the data string. This is then printed out in an orderly format to make counting each container image test result easier.



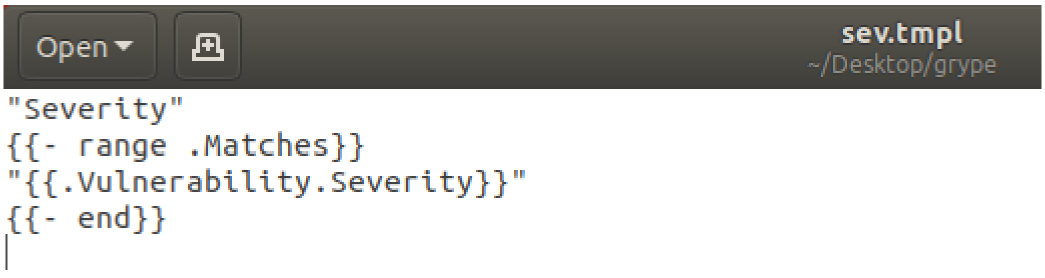
```python
#get reference to the file
file = open(r"C:\Users\Jonat\Desktop\Final Yr Work\Diss\dagdaFiles\drupal.txt")

#read content of file and assign it to a string
data = file.read()

#get number of occurrences of the substring wthin the string
BID = data.count("BID")
RHSA = data.count("RHSA")
RHBA = data.count("RHBA")
CVELOW = data.count('"cvss_access_complexity": "LOW"')
CVEMEDIUM = data.count('"cvss_access_complexity": "MEDIUM"')
CVEHIGH = data.count('"cvss_access_complexity": "HIGH"')
CVECRIT = data.count('"cvss_access_complexity": "CRITICAL"')

#print the number of occurences of each vulnerability
print('Number of occurrences of CVELOW :', CVELOW)
print('Number of occurrences of CVEMEDIUM :', CVEMEDIUM)
print('Number of occurrences of CVEHIGH :', CVEHIGH)
print('Number of occurrences of CVECRIT :', CVECRIT)
print('Number of occurrences of BID :', BID)
print('Number of occurrences of RHSA :', RHSA)
print('Number of occurrences of RHBA :', RHBA)
```

```
C:\Users\Jonat\PycharmProjects\vulnCounter\venv\Scripts\python.exe C:/Users/Jonat/PycharmProjects/vulnCounter/main.py
Number of occurrences of CVELOW : 185
Number of occurrences of CVEMEDIUM : 376
Number of occurrences of CVEHIGH : 128
Number of occurrences of CVECRIT : 0
Number of occurrences of BID : 936
Number of occurrences of RHSA : 0
Number of occurrences of RHBA : 0

Process finished with exit code 0
```

Figure 27, Python code to count occurrences of each type of exploit within a text file.

3.2.5.2 Grype Vulnerability Severity Template

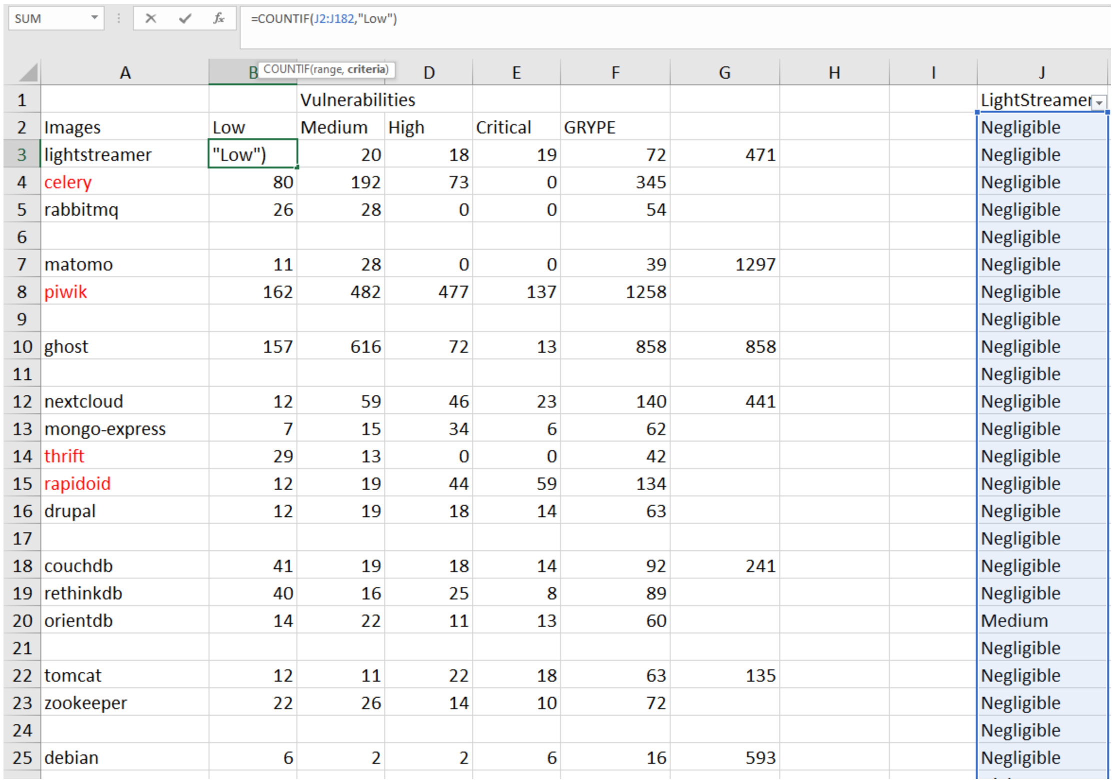
During the Grype testing, it was noted that there would be a design challenge when counting the number of vulnerabilities found within each container image test, just like Dagda. A custom-defined output template that renders CSV data was used during each test only to show the severity of the exploits found to overcome this challenge. Due to the output configuration that Grype provides, the use of a custom-defined output template saved considerable time in the testing phase. A small Go template file was created to suit the project's needs.



```
"Severity"
{{- range .Matches}}
"{{.Vulnerability.Severity}}"
{{- end}}
```

Figure 28, Custom Output Template for Grype

The output from the Grype test using this template was then redirected to a unique text file for each container image. The results within the text file were then loaded into an Excel Spreadsheet so that the results could be analysed. The figure below shows how a simple COUNTIF function counted the imported test results.



SUM =COUNTIF(J2:J182,"Low")

COUNTIF(range, criteria)

	A	B	C	D	E	F	G	H	I	J
1			Vulnerabilities							LightStreamer
2	Images	Low	Medium	High	Critical	GRYPE				Negligible
3	lightstreamer	"Low")	20	18	19	72	471			Negligible
4	celery	80	192	73	0	345				Negligible
5	rabbitmq	26	28	0	0	54				Negligible
6										Negligible
7	matomo	11	28	0	0	39	1297			Negligible
8	piwik	162	482	477	137	1258				Negligible
9										Negligible
10	ghost	157	616	72	13	858	858			Negligible
11										Negligible
12	nextcloud	12	59	46	23	140	441			Negligible
13	mongo-express	7	15	34	6	62				Negligible
14	thrift	29	13	0	0	42				Negligible
15	rapidoid	12	19	44	59	134				Negligible
16	drupal	12	19	18	14	63				Negligible
17										Negligible
18	couchdb	41	19	18	14	92	241			Negligible
19	rethinkdb	40	16	25	8	89				Negligible
20	orientdb	14	22	11	13	60				Medium
21										Negligible
22	tomcat	12	11	22	18	63	135			Negligible
23	zookeeper	22	26	14	10	72				Negligible
24										Negligible
25	debian	6	2	2	6	16	593			Negligible

Figure 29, Excel Spreadsheet COUNTIF Function.

The results from all the vulnerability assessment tools were collectively added to this excel spreadsheet. The study was able to use the results collected within the excel spreadsheet to compare the results.

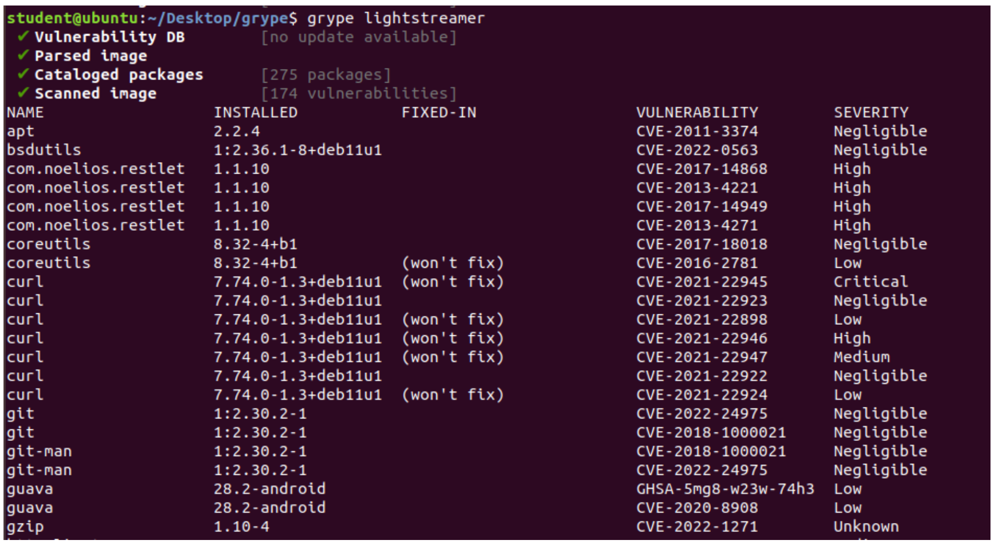
3.2.5.3 Altercations to the planned approach

During the project planning, the plan was to investigate four tools. These included Grype, Trivy, Dagda and Clair. However, during the design process, throughout the configuration of the Clair tool, it was found that the setup was incorrect, and the tool was unable to run. Due to the limited time of the project, a decision was made to only test three of the four initially planned tools due to a lack of time to set up and configure the Clair tool correctly.

3.3 CONTAINER IMAGE TESTING

3.3.1 Grype

Grype outputs CVEs in a table within the terminal when testing container images with no additional configuration. The below figure shows a basic test of a container image.

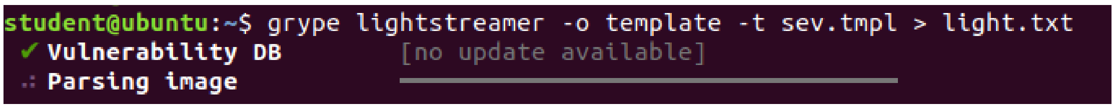


```
student@ubuntu:~/Desktop/grype$ grype lightstreamer
 ✓ Vulnerability DB          [no update available]
 ✓ Parsed image
 ✓ Cataloged packages        [275 packages]
 ✓ Scanned image             [174 vulnerabilities]
NAME                  INSTALLED            FIXED-IN                 VULNERABILITY        SEVERITY
apt                   2.2.4                                         CVE-2011-3374        Negligible
bsdutils              1:2.36.1-8+deb11u1                            CVE-2022-0563        Negligible
com.noelios.restlet   1.1.10                                        CVE-2017-14868       High
com.noelios.restlet   1.1.10                                        CVE-2013-4221        High
com.noelios.restlet   1.1.10                                        CVE-2017-14949       High
com.noelios.restlet   1.1.10                                        CVE-2013-4271        High
coreutils             8.32-4+b1                                     CVE-2017-18018       Negligible
coreutils             8.32-4+b1            (won't fix)              CVE-2016-2781        Low
curl                  7.74.0-1.3+deb11u1   (won't fix)              CVE-2021-22945       Critical
curl                  7.74.0-1.3+deb11u1                            CVE-2021-22923       Negligible
curl                  7.74.0-1.3+deb11u1   (won't fix)              CVE-2021-22898       Low
curl                  7.74.0-1.3+deb11u1   (won't fix)              CVE-2021-22946       High
curl                  7.74.0-1.3+deb11u1   (won't fix)              CVE-2021-22947       Medium
curl                  7.74.0-1.3+deb11u1                            CVE-2021-22922       Negligible
curl                  7.74.0-1.3+deb11u1   (won't fix)              CVE-2021-22924       Low
git                   1:2.30.2-1                                    CVE-2022-24975       Negligible
git                   1:2.30.2-1                                    CVE-2018-1000021     Negligible
git-man               1:2.30.2-1                                    CVE-2018-1000021     Negligible
git-man               1:2.30.2-1                                    CVE-2022-24975       Negligible
guava                 28.2-android                                  GHSA-5mg8-w23w-74h3  Low
guava                 28.2-android                                  CVE-2020-8908        Low
gzip                  1.10-4                                        CVE-2022-1271        Unknown
```

Figure 30, Basic Output from Grype Test

It would be difficult to count the vulnerabilities within this table, so further configuration was required to make counting each CVE easier.

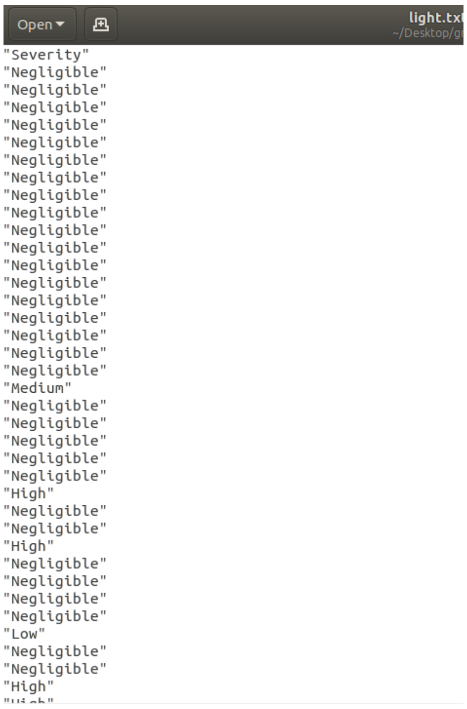
Testing container images using the Grype tool proved quite simple once the output template was created. A simple command could be run to test each container image. The figure below provides an example of a container image being tested for vulnerabilities and exploits.



```
student@ubuntu:~$ grype lightstreamer -o template -t sev.tmpl > light.txt
 ✓ Vulnerability DB          [no update available]
 ⠴ Parsing image
```

In the above command, '-o template' was added to set the output formate to 'template'. Additionally, to specify the path of the template file, '-t sev.tmpl' was added. Finally, '> light.txt' was used to redirect the output of the Grype command to the specified text file.

The figure below shows how the template has been used to only output the severity of each exploit found.

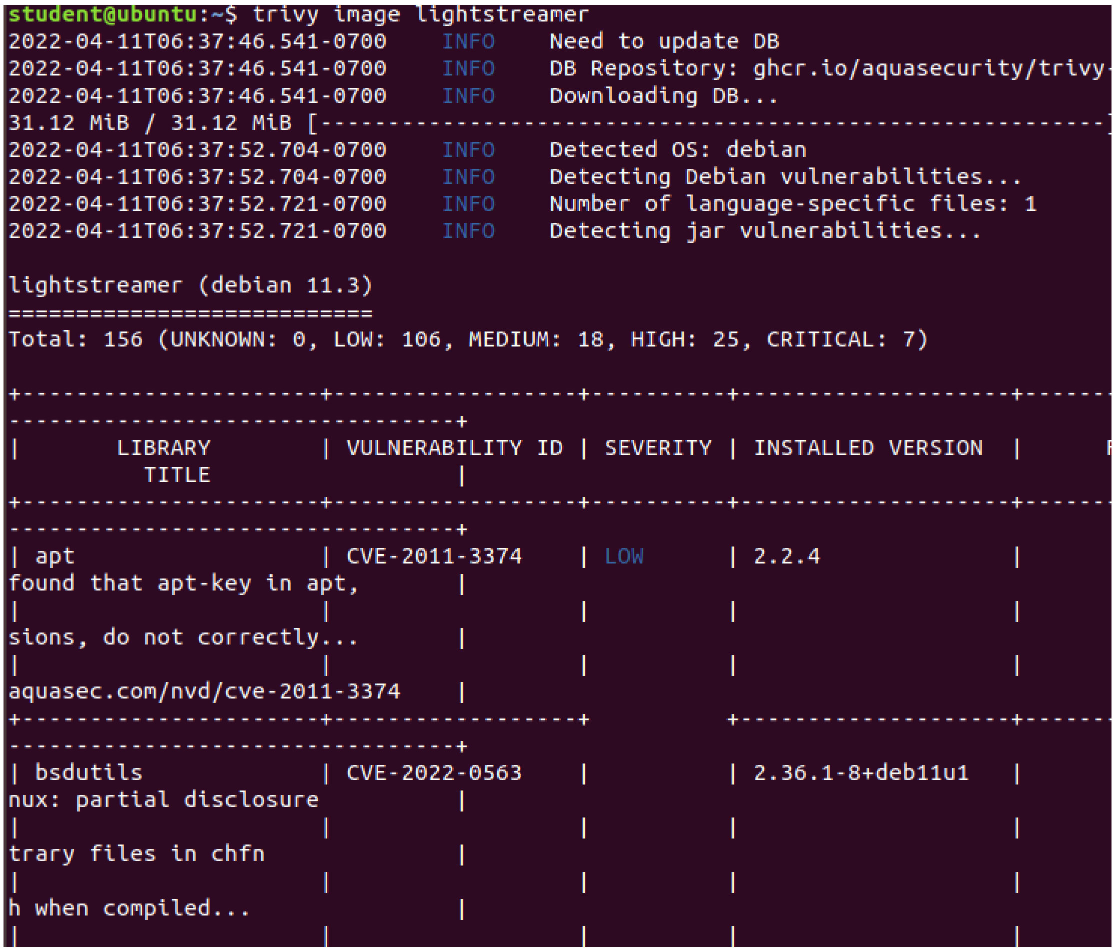


```
"Severity"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Medium"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"High"
"Negligible"
"Negligible"
"High"
"Negligible"
"Negligible"
"Negligible"
"Negligible"
"Low"
"Negligible"
"Negligible"
"High"
```

Figure 31, Grype Template Output

3.3.2 Trivy

Testing images with the Trivy tool proved quite simple and did not require additional configuration once the tool was set up. A simple command is run within the terminal to run a vulnerability test with the Trivy tool. The figure below shows a simple container vulnerability test using the Trivy tool.



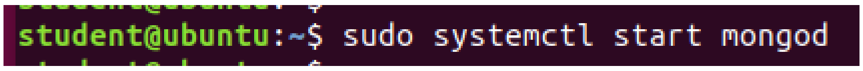
```
student@ubuntu:~$ trivy image lightstreamer
2022-04-11T06:37:46.541-0700    INFO    Need to update DB
2022-04-11T06:37:46.541-0700    INFO    DB Repository: ghcr.io/aquasecurity/trivy
2022-04-11T06:37:46.541-0700    INFO    Downloading DB...
31.12 MiB / 31.12 MiB [--------------------------------------------------------
2022-04-11T06:37:52.704-0700    INFO    Detected OS: debian
2022-04-11T06:37:52.704-0700    INFO    Detecting Debian vulnerabilities...
2022-04-11T06:37:52.721-0700    INFO    Number of language-specific files: 1
2022-04-11T06:37:52.721-0700    INFO    Detecting jar vulnerabilities...

lightstreamer (debian 11.3)
===========================
Total: 156 (UNKNOWN: 0, LOW: 106, MEDIUM: 18, HIGH: 25, CRITICAL: 7)

+-----------------------+------------------+----------+---------------------+------
---------------------------------+
|        LIBRARY        | VULNERABILITY ID | SEVERITY | INSTALLED VERSION   |
          TITLE                  |
+-----------------------+------------------+----------+---------------------+------
---------------------------------+
| apt                   | CVE-2011-3374    | LOW      | 2.2.4               |
found that apt-key in apt,       |
|                       |                  |          |                     |
sions, do not correctly...       |
|                       |                  |          |                     |
aquasec.com/nvd/cve-2011-3374    |
+-----------------------+------------------+          +---------------------+------
---------------------------------+
| bsdutils              | CVE-2022-0563    |          | 2.36.1-8+deb11u1    |
nux: partial disclosure          |
|                       |                  |          |                     |
trary files in chfn              |
|                       |                  |          |                     |
h when compiled...               |
|                       |                  |          |                     |
```

Figure 32, Trivy Image Testing

Trivy did not require any additional configuration as the tool provided a simple total of the CVEs found at the top of the table. Once again, to help analyse the results, the test data was inputted into an Excel spreadsheet.

3.3.3 Dagda

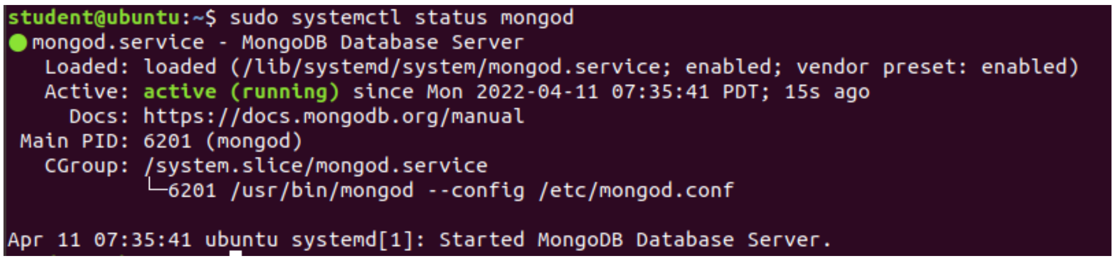
The Dagda tool required a few extra steps and configurations before any testing could be done. This is mainly because the tool requires users to set up and populate their database manually. Firstly, the Mongo Database must be active before the Dagda tool can be run. The figure below shows the command to start up the database service.



```
student@ubuntu:~$ sudo systemctl start mongod
```

Figure 33, Starting MongoDB

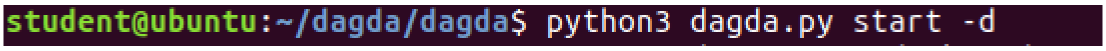
After running the above command, a simple check can be done to check the status of the database to make sure no errors have occurred in the startup of the database. This check is done to help mitigate future errors when running the tool. The figure below shows the database service check.



```
student@ubuntu:~$ sudo systemctl status mongod
● mongod.service - MongoDB Database Server
   Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2022-04-11 07:35:41 PDT; 15s ago
     Docs: https://docs.mongodb.org/manual
 Main PID: 6201 (mongod)
   CGroup: /system.slice/mongod.service
           └─6201 /usr/bin/mongod --config /etc/mongod.conf

Apr 11 07:35:41 ubuntu systemd[1]: Started MongoDB Database Server.
```

Figure 34, Checking the status of the MongoDB

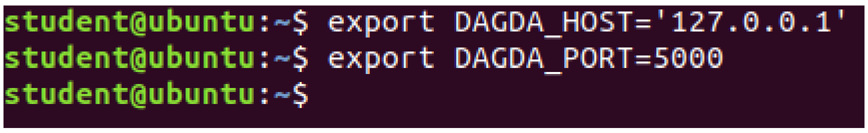
Once we have confirmation on the status of the MongoDB, the Dagda tool can now be run. In a new terminal, after using the 'cd' command to change the directory to the Dagda install locationn, the Dagda tool is run using the command in the figure below.



```
student@ubuntu:~/dagda/dagda$ python3 dagda.py start -d
```

Figure 35, Starting Dagda

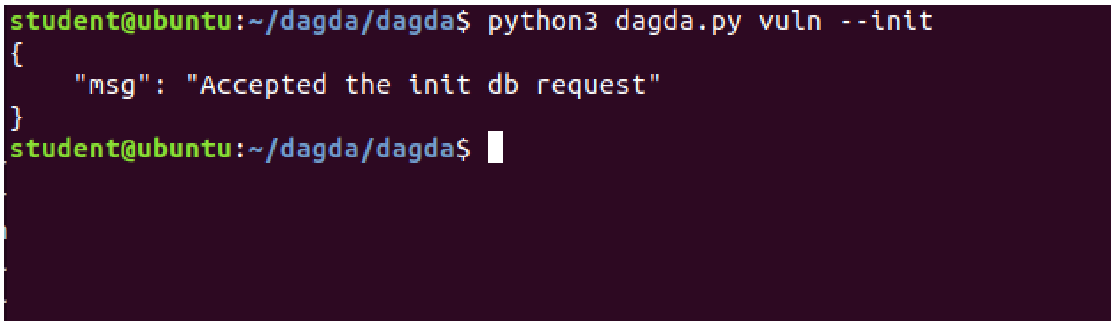
The '-d' was added to the end of the command to enable debugging within the server. This option is added to see what the server is doing when populating the database. Once the server has started, the environment variables must be set before any CLI usage can be done. This is shown in the figure below.



```
student@ubuntu:~$ export DAGDA_HOST='127.0.0.1'
student@ubuntu:~$ export DAGDA_PORT=5000
student@ubuntu:~$
```

Figure 36, Setting environment variable

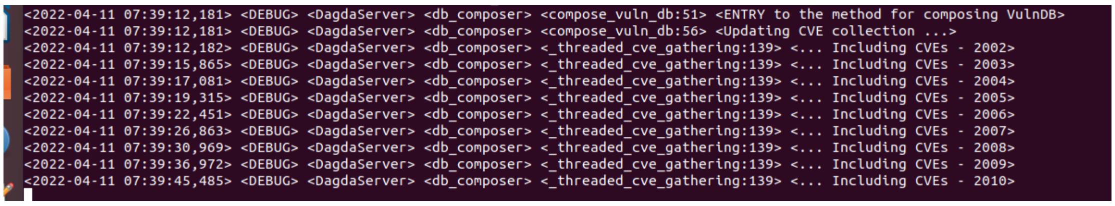
After the environment variables have been set, the database must be populated with the latest vulnerabilities and exploits before any testing occurs. The figure below shows the command to begin the database population.



```
student@ubuntu:~/dagda/dagda$ python3 dagda.py vuln --init
{
    "msg": "Accepted the init db request"
}
student@ubuntu:~/dagda/dagda$
```

Figure 37, Server message showing the acceptance of the initial database population

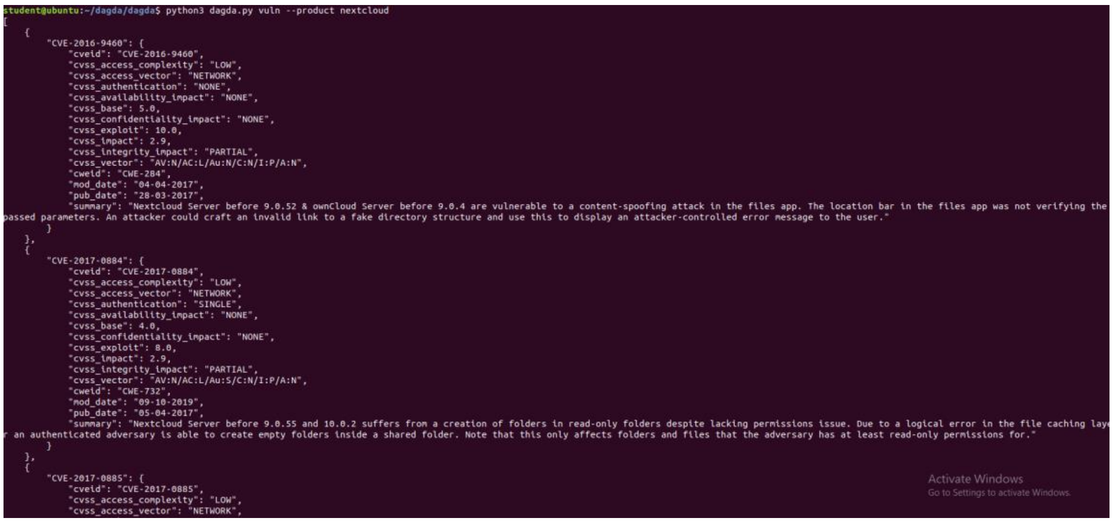
Due to the debug option, the server terminal shows the population of the database in real-time. This is shown in the figure below.



```
<2022-04-11 07:39:12,181> <DEBUG> <DagdaServer> <db_composer> <compose_vuln_db:51> <ENTRY to the method for composing VulnDB>
<2022-04-11 07:39:12,181> <DEBUG> <DagdaServer> <db_composer> <compose_vuln_db:56> <Updating CVE collection ...>
<2022-04-11 07:39:12,182> <DEBUG> <DagdaServer> <db_composer> <_threaded_cve_gathering:139> <... Including CVEs - 2002>
<2022-04-11 07:39:15,865> <DEBUG> <DagdaServer> <db_composer> <_threaded_cve_gathering:139> <... Including CVEs - 2003>
<2022-04-11 07:39:17,081> <DEBUG> <DagdaServer> <db_composer> <_threaded_cve_gathering:139> <... Including CVEs - 2004>
<2022-04-11 07:39:19,315> <DEBUG> <DagdaServer> <db_composer> <_threaded_cve_gathering:139> <... Including CVEs - 2005>
<2022-04-11 07:39:22,451> <DEBUG> <DagdaServer> <db_composer> <_threaded_cve_gathering:139> <... Including CVEs - 2006>
<2022-04-11 07:39:26,863> <DEBUG> <DagdaServer> <db_composer> <_threaded_cve_gathering:139> <... Including CVEs - 2007>
<2022-04-11 07:39:30,969> <DEBUG> <DagdaServer> <db_composer> <_threaded_cve_gathering:139> <... Including CVEs - 2008>
<2022-04-11 07:39:36,972> <DEBUG> <DagdaServer> <db_composer> <_threaded_cve_gathering:139> <... Including CVEs - 2009>
<2022-04-11 07:39:45,485> <DEBUG> <DagdaServer> <db_composer> <_threaded_cve_gathering:139> <... Including CVEs - 2010>
```

Figure 38, Server populating the MongoDB

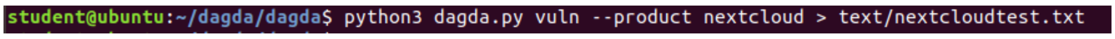
Container image testing can begin once the database has finished populating the server. The figure below shows a simple container image test within the terminal.



```
student@ubuntu:~/dagda/dagda$ python3 dagda.py vuln --product nextcloud
[
    {
        "CVE-2016-9460": {
            "cveid": "CVE-2016-9460",
            "cvss_access_complexity": "LOW",
            "cvss_access_vector": "NETWORK",
            "cvss_authentication": "NONE",
            "cvss_availability_impact": "NONE",
            "cvss_base": 5.0,
            "cvss_confidentiality_impact": "NONE",
            "cvss_exploit": 10.0,
            "cvss_impact": 2.9,
            "cvss_integrity_impact": "PARTIAL",
            "cvss_vector": "AV:N/AC:L/Au:N/C:N/I:P/A:N",
            "cweid": "CWE-284",
            "mod_date": "04-04-2017",
            "pub_date": "28-03-2017",
            "summary": "Nextcloud Server before 9.0.52 & ownCloud Server before 9.0.4 are vulnerable to a content-spoofing attack in the files app. The location bar in the files app was not verifying the
passed parameters. An attacker could craft an invalid link to a fake directory structure and use this to display an attacker-controlled error message to the user."
        }
    },
    {
        "CVE-2017-0884": {
            "cveid": "CVE-2017-0884",
            "cvss_access_complexity": "LOW",
            "cvss_access_vector": "NETWORK",
            "cvss_authentication": "SINGLE",
            "cvss_availability_impact": "NONE",
            "cvss_base": 4.0,
            "cvss_confidentiality_impact": "NONE",
            "cvss_exploit": 8.0,
            "cvss_impact": 2.9,
            "cvss_integrity_impact": "PARTIAL",
            "cvss_vector": "AV:N/AC:L/Au:S/C:N/I:P/A:N",
            "cweid": "CWE-732",
            "mod_date": "09-10-2019",
            "pub_date": "05-04-2017",
            "summary": "Nextcloud Server before 9.0.55 and 10.0.2 suffers from a creation of folders in read-only folders despite lacking permissions issue. Due to a logical error in the file caching laye
r an authenticated adversary is able to create empty folders inside a shared folder. Note that this only affects folders and files that the adversary has at least read-only permissions for."
        }
    },
    {
        "CVE-2017-0885": {
            "cveid": "CVE-2017-0885",
            "cvss_access_complexity": "LOW",
            "cvss_access_vector": "NETWORK",
```

Figure 39, Container Image Test using Dagda

When collecting data from the collection of container image tests, the output of each test was redirected to a text file. This is shown in the figure below.



```
student@ubuntu:~/dagda/dagda$ python3 dagda.py vuln --product nextcloud > text/nextcloudtest.txt
```

Figure 40, Redirecting output to a text file

Each test file created from the collection of container images was then run through the vulnerability counter program. After counting up the exploits found within each image, the data was added to the Excel spreadsheet for analysis.

3.4 Results

This section of the paper will provide an overview of the results obtained from each tool and discuss the comparison of each tool's effectiveness. Images outlined in red are discontinued.

3.4.1 Grype Image Tests

CVE Vulnerabilities

Images	Low	Medium	High	Critical	Total
lightstreamer	15	20	18	19	**72**
celery (DC)	80	192	73	0	**345**
rabbitmq	26	28	0	0	**54**
matomo	11	28	0	0	**39**
piwik (DC)	162	482	477	137	**1258**
ghost	157	616	72	13	**858**
nextcloud	12	59	46	23	**140**
mongo-express	7	15	34	6	**62**
thrift (DC)	29	13	0	0	**42**
rapidoid (DC)	12	19	44	59	**134**
drupal	12	19	18	14	**63**
couchdb	41	19	18	14	**92**
rethinkdb	40	16	25	8	**89**
orientdb	14	22	11	13	**60**
tomcat	12	11	22	18	**63**
zookeeper	22	26	14	10	**72**
debian	6	2	2	6	**16**
centos	94	174	9	0	**277**
ros	121	120	7	1	**249**
ubuntu	30	21	0	0	**51**

Table 3, Grype Results

3.4.2 Trivy Image Tests

CVE Vulnerabilities

Images	Low	Medium	High	Critical	Total
lightstreamer	108	28	23	18	**177**
Celery (DC)	203	164	149	56	**572**
rabbitmq	35	24	0	0	**59**
matomo	282	44	29	14	**369**
piwik (DC)	507	502	481	121	**1611**
Ghost	63	5	4	6	**78**
Nextcloud	339	84	54	18	**495**
mongo-express	0	2	18	0	**20**
thrift (DC)	52	9	0	0	**61**
rapidoid (DC)	80	26	7	8	**121**
drupal	283	44	29	14	**370**
couchdb	131	19	27	8	**185**
rethinkdb	128	19	27	8	**182**
orientdb	81	9	6	8	**104**
tomcat	108	28	23	18	**177**
zookeeper	72	7	4	6	**89**
debian	63	5	4	6	**78**
centos	93	165	8	0	**266**

	CVE Vulnerabilities				
Images	**Low**	**Medium**	**High**	**Critical**	**Total**
ros	144	106	0	0	**250**
ubuntu	35	23	0	0	**58**

Table 4, Trivy Results

3.4.3 Dagda Image Tests

	CVE Vulnerabilities					Bugtraq Exploits	
Images	**Low**	**Medium**	**High**	**Critical**	**CVE Total**	**BID**	**Total**
lightstreamer	0	0	0	0	**0**	0	**0**
Celery (DC)	0	2	0	0	**2**	1	**3**
Rabbitmq	12	9	0	0	**21**	10	**31**
Matomo	11	10	0	0	**21**	0	**21**
piwik (DC)	0	1	0	0	**1**	24	**25**
ghost	3	1	0	0	**4**	25	**29**
nextcloud	34	13	1	0	**48**	16	**64**
mongo-express	3	1	0	0	**4**	725	**729**
thrift (DC)	16	0	0	0	**16**	2	**18**
rapidoid (DC)	0	0	0	0	**0**	0	**0**
drupal	185	376	128	0	**689**	936	**1625**
couchdb	9	8	0	0	**17**	8	**25**
rethinkdb	0	0	0	0	**0**	0	**0**
orientdb	2	3	0	0	**5**	1	**6**
tomcat	110	72	12	0	**194**	169	**363**
zookeeper	5	3	1	0	**9**	3	**12**
debian	0	0	1	0	**1**	5441	**5442**
centos	4	2	0	0	**6**	1317	**1323**
ros	0	1	0	0	**1**	4	**5**
ubuntu	5	3	0	0	**8**	4724	**4732**

Table 5, Dagda Results

3.4.4 Tool Comparison

This section provides a detailed discussion on the comparison of results. Firstly, the most important result from each test is the total amount of exploits found. In this paper, vulnerabilities with negligible risk were not added to the analysis. The figure below shows the total vulnerabilities found per tool.

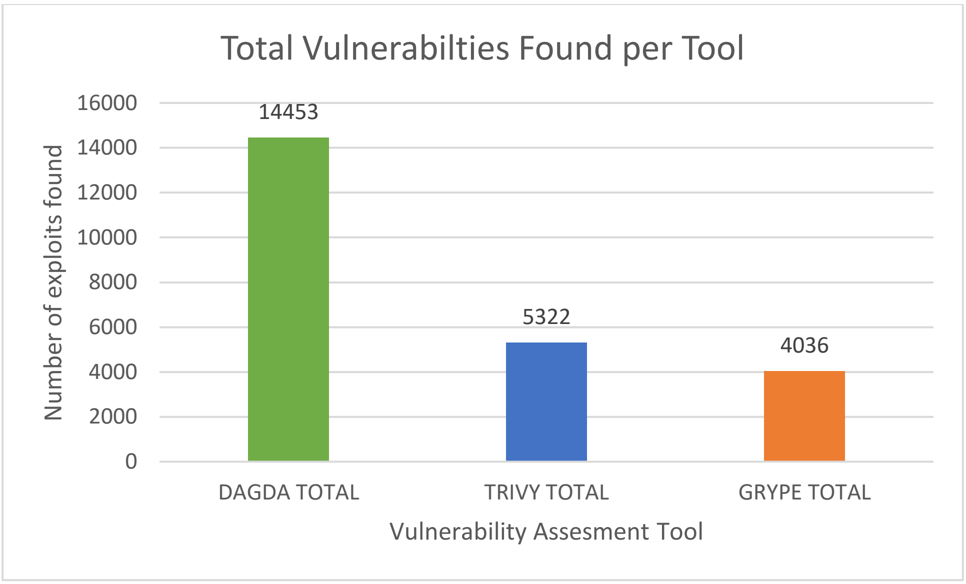


Figure 41, Graph showing total vulnerabilities found per tool

The figure above shows that the Dagda tool was the most effective at finding the most considerable amount of exploits. It was found that Dagda had discovered nearly triple the number of exploits than the other two tools tested within this paper. Dagda was able to find such a large amount of exploits since it could scan for multiple different types of exploits and was not limited solely to CVEs. However, other data suggest that it may not be the most effective at finding high-level CVEs. The figure below shows that Dagda was the worst tool for finding high-risk CVEs.

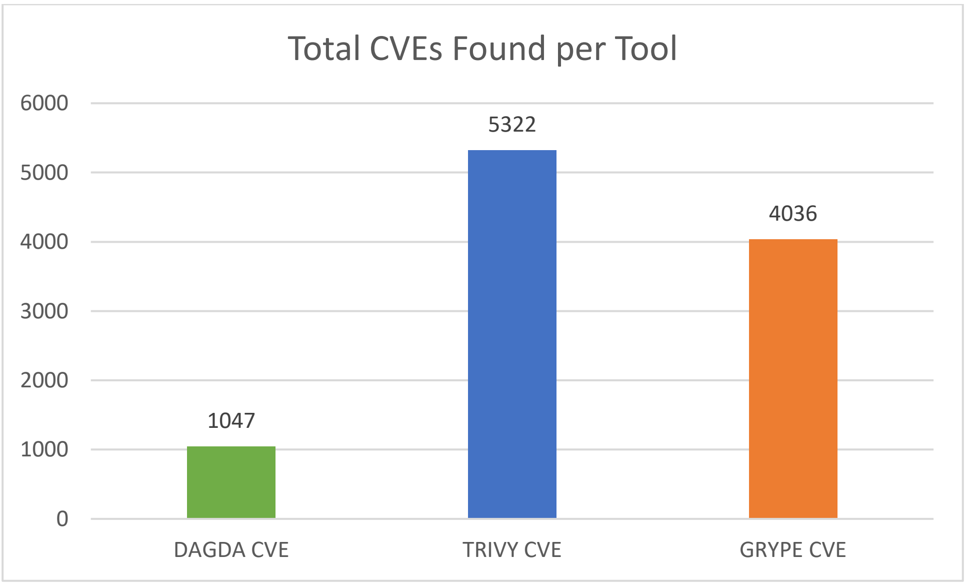


Figure 42, Graph showing the total amount of CVEs found per tool

The figure above shows how effectively each vulnerability assessment tool discovered CVEs within container images. The figure above shows that Trivy and Grype outperformed Dagda when discovering Common Vulnerabilities and Exploits. From the figure above, it is clear that Trivy was the most effective at finding the largest total of CVEs.

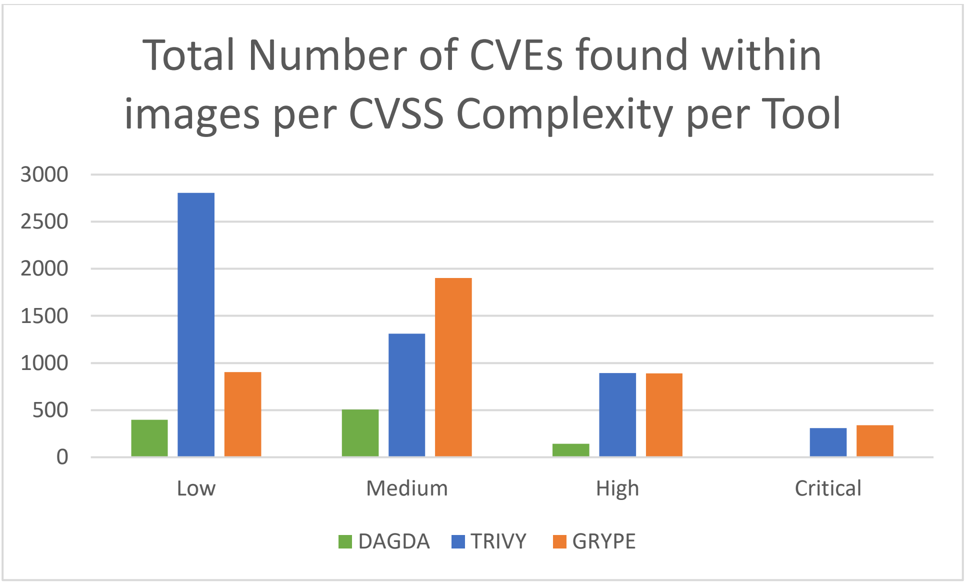


Figure 43, Graph showing the total number of CVEs found per CVSS Complexity per tool

The figure above shows how effectively each tool discovered CVEs within container imagers per CVSS complexity. The figure above shows that Trivy was the best all-around tool for discovering CVEs within the container images tested.

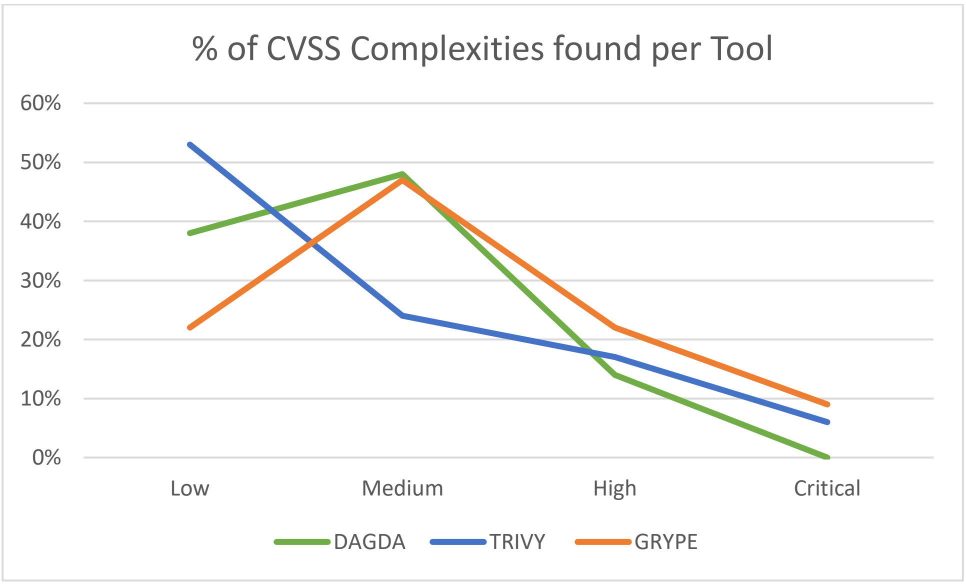


Figure 44, Graph showing the percentage of CVSS complexities found per tool.

The figure above shows the percentage of CVSS complexities found per tool. The figure above shows that the Grype tool was the most effective at finding medium-critical risk vulnerabilities. The figure above shows that Grype specialises in high-risk vulnerability detection compared to Dagda and Trivy. However, the Trivy tool proved the most effective at finding low-risk vulnerabilities.

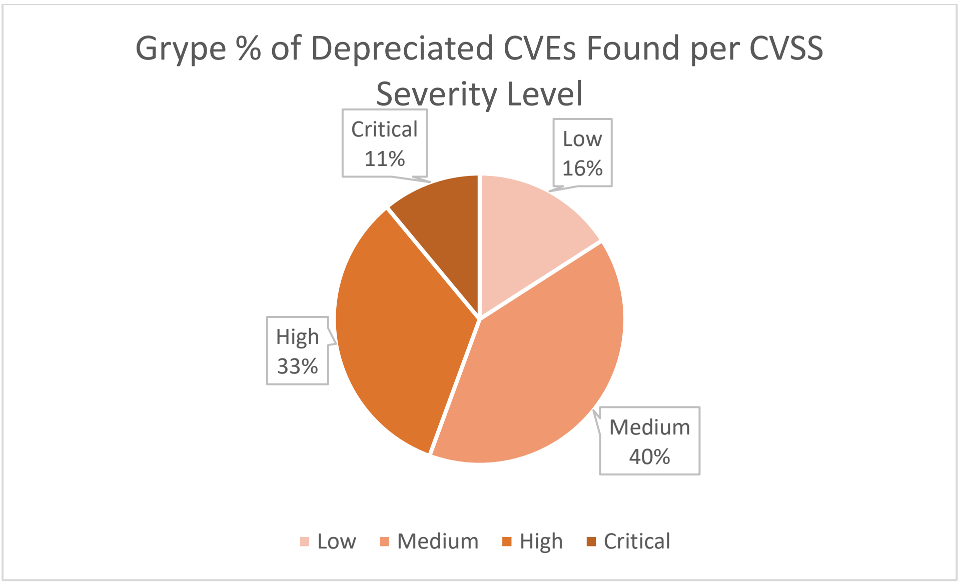


Figure 45, Graph showing Grypes percentage of CVEs found within depreciated images per CVSS complexity

The figure above shows how effective the Grype vulnerability assessment tool was at discovering CVEs within depreciated container images. From the figure above, it is clear that Grype was the most effective at finding 'medium' to 'high' severity level exploits.

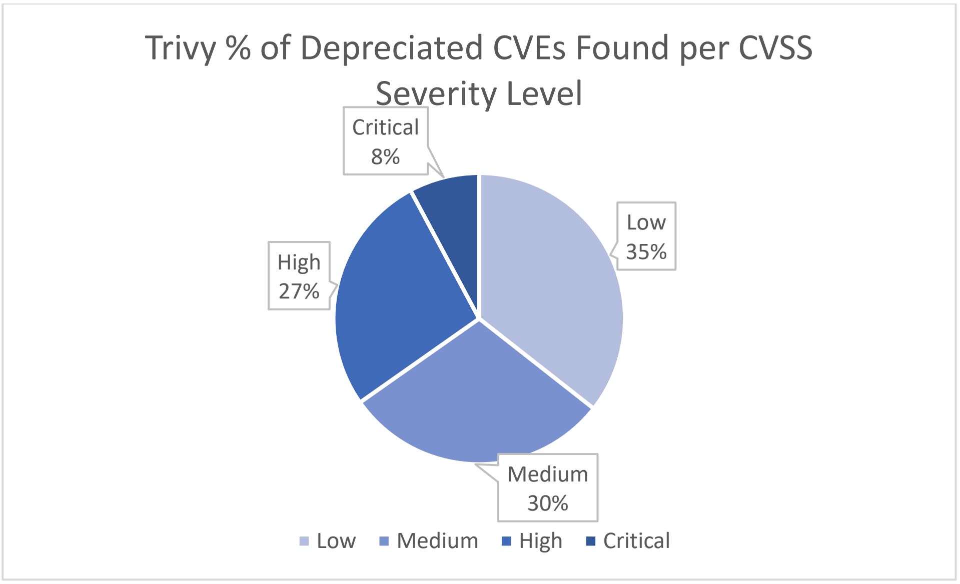


Figure 46, Graph showing the percentage of CVEs found within depreciated images for the Trivy tool

The figure above shows how effective the Trivy tool was at discovering common vulnerabilities and exploits within each CVSS complexity category. The figure above shows that Trivy performed well at finding 'low' to 'high' severity level exploits.

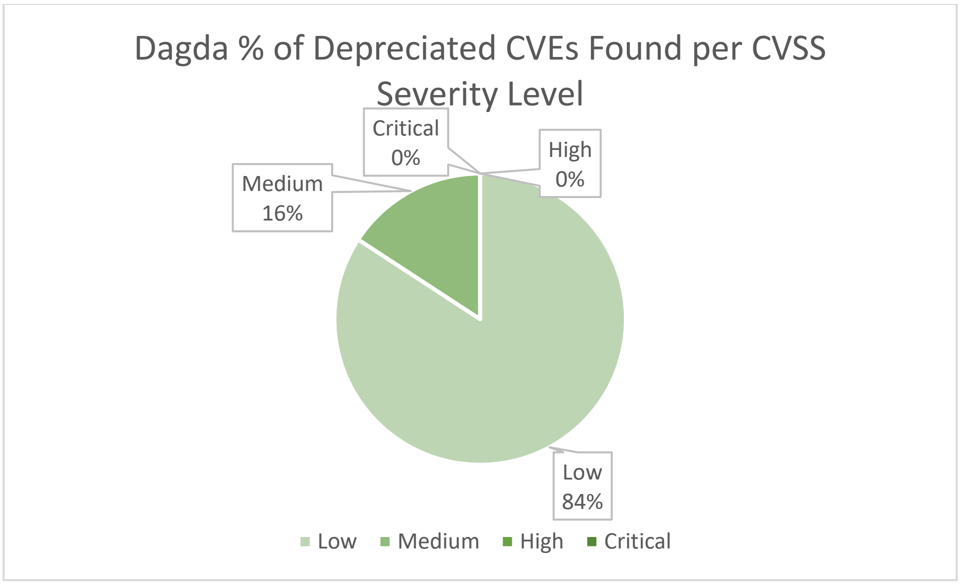


Figure 47, Graph showing Dagda's percentage of depreciated CVEs found per CVSS complexity

The figure above shows how well the Dagda tool discovered CVEs within depreciated images. The figure above shows that all of the CVEs that Dagda found within depreciated container images had a 'medium' or lower severity level. The figure also shows that the majority of the CVEs found had a severity level of 'low'.

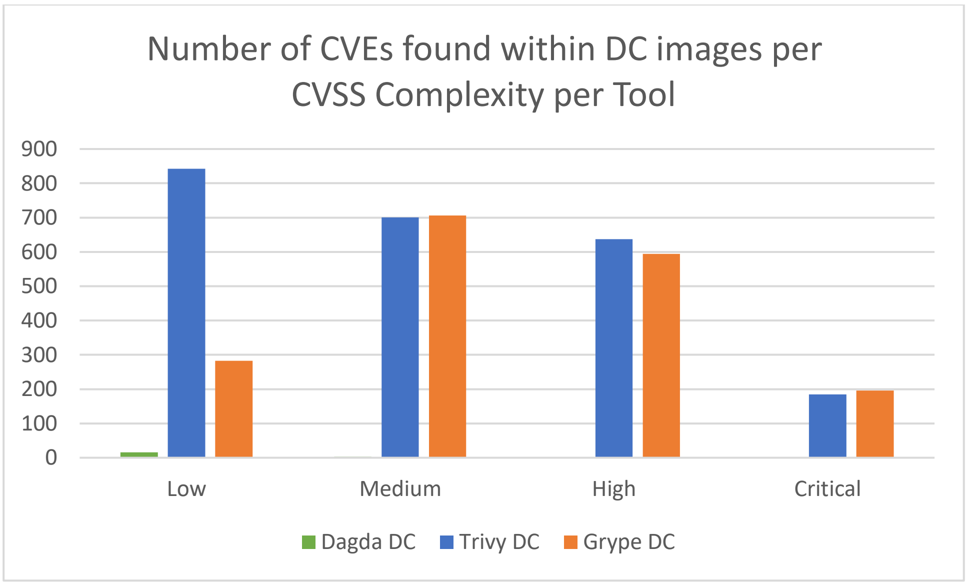


Figure 48, Graph showing the number of CVEs found within depreciated container images per CVSS complexity per tool

The figure above shows how effective each tool was at discovering CVEs in depreciated images within each CVSS complexity. The figure above shows that Trivy outperformed the other two tools and was the most effective at discovering CVEs within depreciated container images.

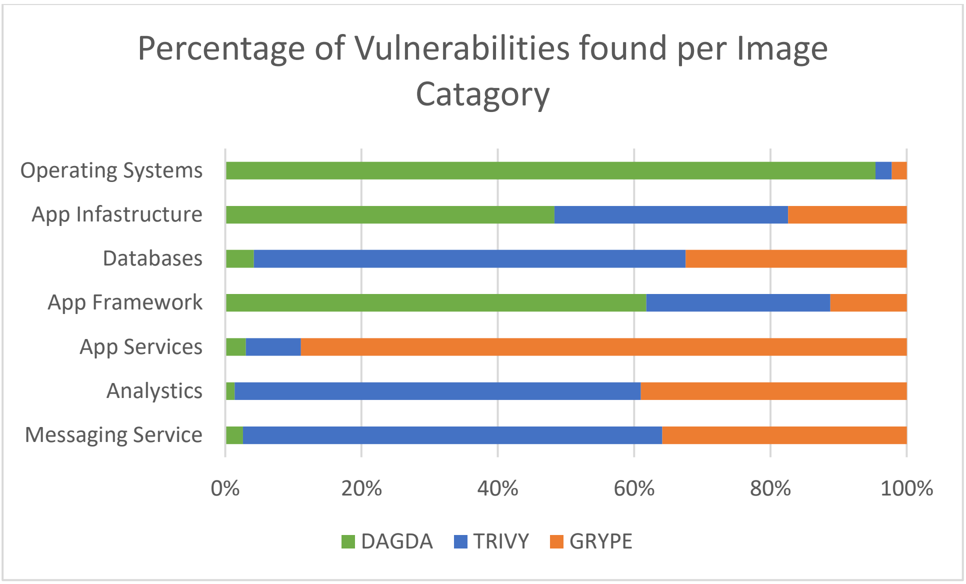


Figure 49, Graph showing percentage of exploits found per image category per tool.

The figure above shows that Dagda was the most effective at finding exploits within Operating System, App Infrastructure and App Framework images. The Trivy tool was the most effective at finding exploits within Database, Analytics and Messaging Service images. Finally, Grype was the most effective at finding exploits within App Service images.

4 Evaluation, Conclusions & Recommendations

4.1 Discussion & Evaluation of Findings

4.1.1 Result Summary

This paper tested three open-source vulnerability assessment tools against twenty container images from the free, open-source Docker Hub repository. After running tests on multiple different image categories, including both active and discontinued images, it was found that Dagda had found the most amount of exploits within the images. Dagda was also the most effective at finding exploits within the operating system, app infrastructure and app framework image categories. Excluding the Bugtraq exploits, the study showed that Trivy was the most effective at finding the total number of CVEs within the container images. Trivy was also very effective at finding exploits within the database, analytics and message service image categories. However, the majority of exploits found within the Trivy tool had a CVSS score of 'Low', meaning that the tool was only effective at finding low-risk vulnerabilities. Finally, Grype was the most effective vulnerability assessment tool for finding medium or high-risk CVEs.

Although finding the largest amount of exploits within an image is important, it is crucial to understand that this alone does not classify a tool as effective. This study has proven that the number of vulnerabilities found may not always show a tool's effectiveness. From Figure 41, it is clear that Trivy had found the most amount of CVEs; however, Figure 42 clearly shows that Trivy was only effective at finding low-risk exploits. The tool's effectiveness ranges due to organisational-defined needs. This may be effective in specific environments where finding low-risk vulnerabilities are the aim. However, most organisations would opt for discovering more severe vulnerabilities, as they pose more risk to their software and information systems.

Testing vulnerable images was also a crucial part of the investigation. Including depreciated container images within the testing phase of the investigation provided the study with a way of understanding the effectiveness of vulnerable container images. The investigation found that Trivy was the most effective at discovering CVEs with vulnerable images. It was found that Trivy discovered the most amount of CVEs within the depreciated container images, discovering 2365 CVEs. Further, the Grype tool was not as efficient as Trivy. However, the Grype tool still managed to discover a considerable 1779 CVEs within depreciated container images.

On the other hand, the Dagda tool was the least efficient and performed poorly when testing depreciated images. With the inclusion of BID exploits, Dagda could only discover 46 exploits within container images that have become depreciated. Finally, both Figure 42 and Figure 47 show that Trivy was the most effective at discovering common vulnerabilities and exploits within depreciated container images. The correlation of results from Figure 47 can also be seen in Figure 42, which suggests that tests on depreciated container images provide no difference in the type of results seen within non-depreciated image tests. These results show that all tools worked just as efficiently on updated images as well as depreciated ones.

Within Figure 48, it can be seen that the vulnerability assessment tools were effective at finding CVEs within specific container image categories. These results are interesting as this suggests that different vulnerability assessment tools are more effective at finding CVEs within different image categories. More research is required in the area, as this could suggest that organisations should use container image specific vulnerability assessment tools to provide the highest level of security towards their pipelines to provide even better organisational dependent security.

This discovery also emphasises that vulnerability assessment tools are more effective within specific areas, which shows that there may not be one effective tool to rule them all. However, the organisations must deploy specific vulnerability assessment tools that coincide with their own organisational needs and dependencies. This can also be seen in Figure 48, as the graph shows how different vulnerability assessment tools are more effective at discovering CVEs within different container image categories.

4.1.2 Effectiveness of Vulnerability Assessment Tools

When installing and configuring the three tools, it was found that Trivy was the easiest of the three. Only a simple curl command was needed to install the tool. The tool also did not require a large amount of RAM or CPU cores to function correctly. It is important to note that although Grype was also a considerably simple install, additional CPU cores and RAM allocation was needed before the tool could function. This additional need for hardware performance and resources may be costly to organisations in a production environment.

On the other hand, the installation and configuration for the Dagda tool were not simple. The ease of use for the Dagda tool is the lowest of the three. This is due to the number of software prerequisites and dependencies required and the additional step of setting up the database separately from the tool itself. These additional steps caused the installation and configuration to take a significant amount of time compared to Grype and Trivy.

Overall, Trivy was found to be the most effective vulnerability assessment tool test in the study. Firstly, Trivy was the easiest of the three tools tested to install. Trivy's only install dependencies required was the curl software. Trivy also required no additional resources for the tool to operate, unlike Grype and Dagda, where additional RAM and CPU cores were required before the tools could function, thus making the tool the cheapest tool of the three to run. After completing all the container image tests, it was also found that Trivy provided all the necessary information, including a tally of the CVEs found within each test and did not require any additional output format configuration or third party programs to count up each vulnerability. The ease of use of the Trivy tool combined with the highest number of CVEs discovered is why Trivy is the most effective vulnerability assessment tool for use within organisational pipelines and deployment/production environments.

4.1.3 Evaluation of Tool Choice

This study tested three open-source static vulnerability assessment tools to discover exploits within container images. The Trivy tool was the most resource-efficient, as this tool required the least resources from the Linux Virtual Machine. The Trivy tool was also efficient at the speed at which it discovered CVEs. The installation of the Trivy tool was also effortless as only once Curl command was used to install. The only thing that let Trivy down was that it lacked the discovery of high-level CVEs compared to the other tools.

Furthermore, the Grype tool required a little more resources from the VM. Without upgrading the VM hardware, the Grype tool failed to test and would freeze the machine. However, the Grype tool was also very speedy at running each container image test and was just as fast as Trivy. The Grype tool was also easily installed with a single Curl command, running within a root terminal. The Grype tool excelled at discovering CVEs with the severity of medium and higher.

On the other hand, the Dagda tool took the longest time out of the three to run individual container image tests. Also, due to the extra addition of a locally created database to hold CVEs and other known exploits, Dagda required the most system resources to run the tests. Having known this, organisations choosing to use the Dagda tool within their pipeline will require additional setup and resources, making the Dagda tool more costly to run in production environments than the other two tools.

Overall this project recommends that organisations use a combination of static analysis tools in conjunction with dynamic vulnerability assessment tools to provide the most security to company pipelines and container technology. As previous research has stated, the concern of both high and low-level CVEs, it is essential that organisations aim to prevent CVEs no matter the severity. From the research and analysis of the investigation, this study recommends organisations use a tool such as Trivy, which is effective at finding low-level CVEs, in combination with a tool such as Grype, which is effective at finding high-level CVEs. Multiple static vulnerability assessment tools should be implemented within organisational pipelines in order to provide the highest level of protection and exploit discovery at a static level. The study also recommends that organisations use the static vulnerability assessment tools in conjunction with dynamic assessment tools to provide the highest container image security within their CI/CD pipelines.

4.2 Evaluation of the Project Process

4.2.1 Achievement of Objectives

This section of the project aims to deliver a complete comparison between the aims and objectives and the work done within the investigative study.

4.2.1.1 Aims

"To investigate container security controls from a popular security framework and compare. "

This aim has been met as the study has looked at multiple use cases for container technology and provided a detailed discussion on the importance of secure containers. The study has also provided a detailed examination of relevant security controls from the popular NIST security framework alongside an analysis of CI/CD pipelines and the role of vulnerability assessment tools within organisational pipelines. These can be found in **Chapters 2.1 - 2.4.4.**

"Evaluate the effectiveness of open-source vulnerability assessment tools against container images."

This aim has been met as the investigative study has looked at what makes a vulnerability tool effective and included relevant tool comparisons from the container image tests done during the investigation. The tool comparison section found within **Chapter 3.4.4** is used to provide clear references within the result summary. The results found within the study in conjunction with the research have provided great insight into the effectiveness of vulnerability assessment tools and have posed questions to organisations if they should be using more container image specific vulnerability assessment tools. This aim has been within **Chapters 3.4 - 4.1.**

4.2.1.2 Objectives

1. **Review previously published similar works**

This objective has been met as the study has established the importance of container security and analysed publications, reviewing security controls within containerised application development and deployment. Multiple research studies have found a lack of awareness against testing static vulnerability assessment tools; therefore was the reason behind this investigative study. **Chapter 2** shows how this objective has been met with literature regarding container security publications and studies on vulnerability assessment tools.

2. **Establish the uses of containers in large-scale enterprise workloads**

This objective has been met as the study has produced research into the area of container use cases. This objective aimed to establish an understanding of the purpose of containers within enterprise workloads. **Chapter 2.2** shows how this objective has been met.

3. **Research tools and policies that provide security to containers and infrastructure**

This objective has been met as the investigative study has established a popular security control framework, including research on the role of security controls. The study has also provided detailed research into open-source vulnerability tools. This objective has been met within **Chapters 2.3-2.4** and **3.1**.

4. **Perform in-depth analysis of container security, including security controls and features**

This objective has been met as the study has provided an in-depth analysis of the importance of container security, including recent breaches alongside the role of security controls. The study has also introduced an in-depth analysis of the NIST security control framework and thoroughly examined and analysed relevant security controls found within NIST publications. Furthermore, this study also provided a detailed discussion on CI/CD pipelines and the role of vulnerability assessment tools with literature to support the research. How this objective has been met can be seen throughout **Chapters 2.3** and **2.4**. This objective was crucial to the investigation to provide a strong foundation in container technology security and lay a basis for the reader to understand the importance of this investigative study.

5. **Plan, set up and configure container vulnerability tools**

This objective has been met, as seen within the design **Chapters 3.1 – 3.2**, where the vulnerability assessment tools planning, setup and configurations were discussed. This objective was to ensure the vulnerability assessment tools could run smoothly and was configured to run tests on the container images. Although this objective was met, it was only met to some extent as the investigative study only managed to set up and configure three of the four planned vulnerability assessment tools. After finishing the configuration of the three tools, completing the setup and configuration of the final planned tool, Clair, was simply out of the scope of the study and could not be completed within the time frame of the project plan.

6. **Test popular container images for OS and non-OS vulnerabilities**

This objective has been met as the study has provided a range of tests using multiple vulnerability assessment tools in order to test container images for both OS and non-OS vulnerabilities and exploits. The study underwent twenty image tests per vulnerability assessment tool, which tested each container image for operating system and non-operating system library vulnerabilities. **Chapter 3.3** through to **Chapter 3.4** shows this objective being met. Throughout t

7. **Provide an in-depth analysis of the number of vulnerabilities in container images for each tool used**

This objective has been met as the study presented an evaluation found within **Chapter 4** that included a result summary. The summary included detailed discussions and an in-depth analysis of the results found in **Chapter 3.4**, including references to the vulnerability assessment tool comparison found in **Chapter 3.4.4.**

8. **Evaluate the effectiveness of vulnerability tools**

The study has provided an evaluation of the effectiveness of each vulnerability assessment tool within different areas of the investigation. The study looked at the effectiveness of discovering CVEs, CVSS Complexities, and CVSS Severity Levels and looked into the effectiveness of each tool when put against different container image categories. This objective has been met and can be seen in **Chapters 4.1.2 - 4.1.3.**

9. **Propose a secure container policy that can be applied to safeguard the use of containers by enterprises**

This objective has been within **Chapters 4.3.2 - 4.3.2.1.** This study has used the current test results alongside the research on existing NIST publications to form container security best practices for optimal security controls within the life cycles of containerised applications. The purpose of this objective was to provide developers and organisations with a secure reccomendation/policy to safeguard their use of containers.

4.2.2 Project Challenges

During this investigation, issues were faced when testing container images for vulnerabilities with the Grype tool. The Grype tool was resource-intensive, and the Virtual Machine froze upon initially running the tests. The Virtual Machine required additional resources to be allocated for the Grype and Dagda tool not to freeze.

Installing and configuring the vulnerability assessment tools was new knowledge. Configuring the connections between the SSH client to Git hub alongside managing internal local databases to run with CLI software was all new technology which was a hurdle in the investigation to overcome. Learning how to manage and configure each vulnerability assessment tool alongside working with container images was challenging and required months of research to understand how to operate each tool. On the other hand, collecting the data was another challenge that required some time to overcome. Due to each test returning thousands of vulnerabilities and exploits, a solution was required to count the CVEs from each test. As a result, a python program was created to speed up the counting of the CVEs discovered.

4.2.3 Skills Learnt

Throughout the project, multiple skills were learnt along the way. Firstly, when working alongside CLI software, an SSH connection was required to install most of the tools and dependencies within the study. This was the first lesson learnt throughout the study, as previous work on SSH client connection was scarce. SSH connection, setup, and configuration skills were picked up throughout the investigative study.

Another lesson learnt throughout the study was the use of local database configuration. As previous work had only required online remote databases, this was a new area that had to be learnt and picked up. One thing that differed from the use of remote databases was that the local database was configured through a CLI. This was also a new process as previous work with remote databases was set up using an online user interface.

Another skill learned during this investigative study was Excel data presentation. Working with a large amount of data, research was done on how best to collate and present the findings. This included the use of Excel Formulas and graph arrangement.

When collating the image analysis tests data, challenges were faced when counting the CVEs from the test results. Problem-solving skills were learnt as a Python program was coded to solve the issue of collecting the total amount of vulnerabilities found within each container image test.

While working through the investigative study, project management skills have been developed. Managing the completion and planning of a large, high-quality piece of work was a new skill required to complete this study. The management of the project involved skills such as organisation, time management and perseverance.

4.2.4 Project Improvements

If this investigative study was to be repeated, this report recommends that more vulnerability assessment tools be chosen as this would broaden the test results. Increasing the number of tools tested would provide organisations and enterprises with a greater understanding of vulnerability assessment tools and allow developers to have more precise knowledge of the pros and cons of each vulnerability assessment tool. In cases where organisational needs require specific tests for certain severity level CVEs, increasing the number of vulnerability assessment tools tested would also prove helpful.

This report also suggests that a more considerable amount of container images tested would also provide the study with a greater understanding of the effectiveness of each tool against container images. Increasing the number of container images tested would provide the study with a greater sample size and a better understanding of each tool's effectiveness. This improvement would also provide organisations and developers with a better understanding of which tool is better specialised at discovering vulnerabilities within specific image categories. This information would be helpful to organisations as they could identify which vulnerability assessment tool would be best suited to their container image development and deployment. Doing so would provide better security to organisational pipelines by introducing container image-specific vulnerability assessment tools.

4.2.5 Alternative Project Plans

Given the benefit of hindsight, additional container images could be provided as more container images for each tool to test would show a fairer and more accurate result. As the time given for this investigation was limited, only a certain amount of images could be tested before the allotted time was up. This report also suggests that further research into static analysis tools should opt for a larger amount of testing tools to broaden the study results.

4.3 Conclusion & Recommendations

4.3.1 Conclusion

To conclude this investigative study, the growing popularity of containerised application use has sparked concern about the use of container images within cloud-native environments. It brought an increasing interest into container technology security. This paper aimed to explain the importance of container security with literature to show relevant security controls that organisations must follow to improve container technology security and protect their CI/CD pipelines. From the research done, it was found that vulnerability assessment tools play a crucial part in the process of securing organisational pipelines.

Research has been sparse on effective vulnerability assessment tools for static image analysis. As containerised applications are a relatively new technology, there is a lack of research papers and literature to review. The existing literature only provided insight into the dynamic analysis of container images. Due to a lack of research into static container image analysis, this paper aimed to investigate the effectiveness of open-source static vulnerability assessment tools. This report aimed to use the results from the investigation to provide organisations and developers with a greater insight into static container image analysis and in-depth analysis of the effectiveness of vulnerability assessment tools.

It was found that Trivy was the overall most effective vulnerability assessment tool during the evaluation of findings. The Trivy tool was the easiest to set up out of the three tools tested and discovered the highest number of CVEs. However, the findings from the investigations posed the question, should organisations only use one type of static and dynamic tool for securing their pipeline? The research from the study showed that vulnerability assessment tools were more effective at finding CVEs within specific container image categories. These findings suggest that organisations should use specific vulnerability assessment tools for certain categories of containerised app development.

Nevertheless, further research is required into using vulnerability assessment tools against specific container image categories before any particular judgement can be made. This investigative study also found that the vulnerability tools were just as effective at discovering CVEs within depreciated images than actively updated images, as their results did not differ. It was found that through both tests, the tools found a similar percentage of CVE complexities. During the project evaluation, it was found that if the project were to be repeated, an increase in the vulnerability assessment tools tested would provide the study with a broader set of results. It would also help organisations whose needs require specific tests for CVEs with a certain CVSS severity level.

4.3.2 Recommendations

Past research has shown that static vulnerability tests alone are not enough to be able to mitigate malware security risks within container images (Binnie, 2018). As a result of this, this study recommends that organisations use dynamic threat analysis tools in conjunction with static vulnerability assessment tools. Tools such as Jfrog Xray can run potentially vulnerable/poisoned images within a safe and isolated environment in order to run dynamic vulnerability tests. Organisations must implement dynamic tests within their pipeline as some malware can escape static tests by only activating once the container is deployed. Organisations that implement SCA tools that can be used to prevent the images from being pushed into production environments in conjunction with static vulnerability assessment tools within their pipeline can prevent vulnerable images from being pushed to repositories and deployed in production environments.

This study also recommends that security control frameworks, such as the NIST 800-53, should be utilised within organisational pipelines. The introduction of security controls and policies within each stage of a company's CI/CD pipeline can significantly reduce the risk of vulnerabilities with their container image production and deployment.

Further research and testing into securing CI/CD pipelines would prove helpful to developers and organisations working with containers. As past research has shown that container CI/CD pipelines require more security than traditional software development, further research into container pipelines would help the industry develop a deeper understanding of the importance of container security. Further testing into the effectiveness of dynamic vulnerability tools is recommended, as container adoption is growing exponentially within organisations. Further research into container security tools will hopefully provide organisations with a greater understanding of how to protect their data and information systems.

This study also recommends further research into static versus dynamic vulnerability assessment tools to provide insight into the effectiveness of each test. As some vulnerability assessment tool provides options for both static and dynamic tests, further research into this area would provide organisations with clarification on tools that specialises in a particular type of image scan versus tools that offer both static and dynamic image testing.

This study showed that different vulnerability assessment tools were more effective within specific image categories. Further research into the effectiveness of vulnerability assessment tools on specific container image categories is recommended. Further research into this area would help determine if implementing container image-specific vulnerability assessment tools would be more secure than using a singular favoured image analysis tool.

4.3.2.1 Secure Policy to Safeguard Container Use

4.3.2.1.1 Important Security Controls

Although there are many benefits to adopting containerised applications within organisational environments, many security factors must be addressed to mitigate the possible security risks accompanying container image use. Firstly, this study has found that it would be beneficial for organisations to follow a security control framework in order to secure the development and deployment of containers. A popular choice this study recommends is the NIST security control framework. Within this study, security controls relevant to container technology security were analysed, and from this analysis, this paper suggests that organisations implement Access Control policies. By introducing MAC policies, organisations can increase security within their pipelines. AC security policies were among the most crucial security controls for securing organisational infrastructure and pipelines from the security framework analysis. Without organisational compliance towards MAC policies, container images may be at high risk of runtime exploitation and system-wide compromises.

4.3.2.1.2 Organisational Best Practices

Secondly, this study found that it would benefit organisations to implement multiple vulnerability assessment tools to scan for static and dynamic images. During the investigative research, it was found that not all vulnerabilities and exploits can be found within static vulnerability assessment tools. As a result of this, organisations must implement both static and dynamic vulnerability assessment tools within their pipelines to safeguard the use of container images within their production environments. The findings of this investigative study have also shown that organisations could benefit from running multiple vulnerability assessment tools per static/dynamic test. The reason behind this is that the investigation showed that different vulnerability assessment tools were more effective at discovering specific CVE severity levels. By introducing CVE severity specific vulnerability assessment tools, organisations will be able to discover a broader range of CVEs within each image. This investigative study also found that different vulnerability assessment tools were more effective at discovering CVEs within different container image categories. As a result, it could be beneficial to organisations to run container image analysis using a vulnerability assessment tool specific to the container image category.

5 REFERENCES

AppDynamics, 2022. *Why DevOps is Important.* [Online]
Available at: https://www.appdynamics.com/topics/why-devops-is-important
[Accessed 7 March 2022].

AVI Networks, 2022. *Microservices And Containers.* [Online]
Available at: https://avinetworks.com/what-are-microservices-and-containers/
[Accessed 16 January 2022].

Barua, H., 2020. *Half of 4 Million Public Docker Hub Images Found to Have Critical Vulnerabilities.* [Online]
Available at: https://www.infoq.com/news/2020/12/dockerhub-image-vulnerabilities/
[Accessed 18 December 2021].

Bélair, M., Laniepce, S. & Menaud, J.-M., 2019. *Leveraging Kernel Security Mechanisms to Improve Container Security: A Survey,* Canterbury: Association for Computing Machinery.

Binnie, C., 2018. Auditing Docker Containers in a DevOps Environment. *ADMIN Real World AWS,* Issue 43, p. 100.

Brady, K., Moon, S., Nguyen, T. & Coffman, J., 2020. *Docker Container Security in Cloud Computing,* Las Vegas: IEEE.

Czikó, Z., 2019. *Management and Vulnerability Scanning of Docker Containers in an Embedded Enviroment,* Budapest: Budapest University of Technology and Economics.

Duncan, B., Lee, Y. & Olmsted, A., 2018. *Cloud Computing 2018 Proceedings of the Ninth International Conference on Cloud Computing, Grids, and Virtualization,* Barcelona: IARIA.

García, V. H., 2021. *NIST 800-53 compliance for containers and Kubernetes.* [Online]
Available at: https://sysdig.com/blog/nist-800-53-compliance/
[Accessed 31 January 2022].

Gartner, 2020. *Gartner Forecasts Strong Revenue Growth for Global Container Management Software and Services Through 2024.* [Online]
Available at: https://www.gartner.com/en/newsroom/press-releases/2020-06-25-gartner-forecasts-strong-revenue-growth-for-global-co
[Accessed 15 January 2022].

Github, 2022. *Trivy.* [Online]
Available at: https://github.com/aquasecurity/trivy
[Accessed 14 Febuary 2022].

Haymore, A., 2022. *10 real-world stories of how we've compromised CI/CD pipelines.* [Online]
Available at: https://research.nccgroup.com/2022/01/13/10-real-world-stories-of-how-weve-compromised-ci-cd-pipelines/
[Accessed 15 Febuary 2022].

Khandelwal, S., 2017. *It's 3 Billion! Yes, Every Single Yahoo Account Was Hacked In 2013 Data Breach.* [Online]
Available at: https://thehackernews.com/2017/10/yahoo-email-hacked.html
[Accessed 16 December 2021].

National Cyber Security Centre, 2021. *Defending software build pipelines from malicious attack.* [Online]
Available at: https://www.ncsc.gov.uk/blog-post/defending-software-build-pipelines-from-malicious-attack
[Accessed 20 January 2022].

National Institue of Standards and Technology, 2017. *Application Container Security Guide,* s.l.: U.S. Department of Commerce.

National Institute of Standards and Technology, 2017. *Application Container Security Guide,* s.l.: U.S. Department of Commerce.

NIST, 2020. *Security and Privacy Controls for Information Systems and Organizations.* [Online]
Available at: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf
[Accessed 9 March 2022].

North Carolina Department of Information Technology, 2022. *Assesment, Authorization and Monitoring Policy.* [Online]
Available at: https://it.nc.gov/media/1751/open
[Accessed 8 March 2022].

Peacock, J., 2022. *NIST SP 800-53 Control Families Explained.* [Online]
Available at: https://www.cybersaint.io/blog/nist-800-53-control-families
[Accessed 27 01 2022].

Red Hat, 2018. *What is CI/CD?.* [Online]
Available at: https://www.redhat.com/en/topics/devops/what-is-ci-cd
[Accessed 12 Febuary 2022].

Rice, L., 2021. *Vulnerability Scanning for Kubernetes Applications: Why and How.* [Online]
Available at: https://blog.aquasec.com/kubernetes-vulnerability-scanning
[Accessed 25 Febuary 2022].

Scroxton, A., 2021. *Low-complexity CVEs a growing concern.* [Online]
Available at: https://www.computerweekly.com/news/252496201/Low-complexity-

CVEs-a-growing-concern
[Accessed 15 March 2022].

SecureCloudDB, 2021. *4 Major Breaches in 2020 and How They Could Have Been Mitigated (Part 1).* [Online]
Available at: https://www.securecloudb.com/blog/4-major-breaches-in-2020-part-1-estee-lauder#:~:text=Half%20a%20Billion%20Records%20Exposed,was%20discovered%20in%20January%202020.
[Accessed 23 Febuary 2022].

Souppaya, M., Morello, J. & Scarfone, K., 2017. *NIST Special Publication 800-190,* Maryland: NIST.

Sultan, S., Ahmad, I. & Dimitriou, T., 2019. *Container Security: Issues, Challenges, and the Road Ahead,* s.l.: IEEE.

Sysdig, 2022. *CI/CD Security: Securing Your CI/CD Pipeline.* [Online]
Available at: https://sysdig.com/learn-cloud-native/container-security/cicd-pipeline/
[Accessed 21 Febuary 2022].

Trend Micro, 2019. *Docker Hub Repository Suffers Data Breach, 190,000 Users Potentially Affected.* [Online]
Available at: https://www.trendmicro.com/vinfo/es/security/news/cybercrime-and-digital-threats/docker-hub-repository-suffers-data-breach-190-000-users-potentially-affected
[Accessed 17 December 2021].

Trend Micro, 2021. *Top 10 AWS Security Misconfiguration.* [Online]
Available at: https://www.trendmicro.com/en_hk/devops/21/k/top-10-aws-security-misconfigurations.html
[Accessed 23 Febuary 2022].

VMware, 2021. *AC-6 LEAST PRIVILEGE.* [Online]
Available at: https://docs.pivotal.io/nist/ac/ac-6.html
[Accessed 10th January 2022].

VMWare, 2022. *AU-2 AUDIT EVENTS.* [Online]
Available at: https://docs.pivotal.io/nist/au/au-2.html
[Accessed 2 March 2022].

VMWare, 2022. *CM-3 CONFIGURATION CHANGE CONTROL.* [Online]
Available at: https://docs.pivotal.io/nist/cm/cm-3.html
[Accessed 3 March 2022].

Warrier, A., 2020. *Containers vs Virtual Machines (VMs).* [Online]
Available at: https://www.eginnovations.com/blog/containers-vs-vms/
[Accessed 18 January 2022].

6 APPENDICES

6.1 APPENDIX A – TERMS OF REFERENCE DOCUMENT

Project Terms of Reference

Module:	KV6003: Individual Computing Project
Full name:	Jonathan Lee Hill
Student ID:	W18015151
Course:	Computer Networks and Cyber Security
Project Title:	Investigation into container security and the effectiveness of security controls
Supervisor:	Rafay Ansari
2nd marker:	Xiaomin Chen
Project type:	Investigative Project

Investigation into container security and the effectiveness of security controls

Background to Project

Containers are growing in popularity in cloud-native development over the use of Virtual Machines (VM's) because they are lightweight, have faster boots times and require less memory space. However, it is essential to understand the security of containers. With the wide adoption of container-based applications and systems alongside the introduction of DevSecOps, it has become apparent that container security is an important aspect of container app development.

Recent work on the current issues and challenges of container security has shown the importance of container security and the possible attacks and vulnerabilities. The report has proposed future research directions in the hopes of spawning further research in the area. (Sultan, et al., 2019). It was mentioned that further research into the usability of vulnerability assessment tools would provide tremendous help to developers and companies.

Previous work has shown that security controls must be introduced to secure the use of container-based virtualisation (National Institue of Standards and Technology, 2017). In September 2017, the National Institute of Standards and Technology (NIST) published the first guide on application container security. This publication discussed security threats, recommendations and countermeasures to safeguard the use of application containers.

The idea for this project came from an interest in cloud computing and virtual machines. After researching the two topics, an upcoming popular alternative to virtual machines was discovered that companies are adopting in development.

This project will be helpful to developers and companies who are using containers as a lightweight alternative to virtual machines. This project aims to provide an in-depth analysis of container security and explore the effectiveness of the current security controls.

Within the study, multiple vulnerability assessment tools will be used to provide an insight into the effectiveness of each tool and show the risks of online docker images.

Proposed Work

Areas of Investigation

This project aims to investigate container security and to determine the effectiveness of security controls.

Topics of literature review

- NIST Guidelines

    - The NIST Guidelines provide controls to strengthen the security of IT systems. Some of the controls mentioned in the guide can be used in the development of container applications. During the container development lifecycle, these controls can be implemented to improve security.
- Container Security And Usage Reports
    - Previous work on container security and its usage will be helpful to this report when establishing the importance of container security. The reports will provide an insight into enterprise workloads and provide a strong foundation for the in-depth analysis of container security, including the security controls and features.
    - Example Case uses:
        - Microservices
        - DevOps
        - Hybrid and multi-cloud environments
        - Application modernisation
            - Lift and shift cloud migration

Research

- Security Controls
    - This project section will be investigating a cybersecurity control framework such as CSA's Cloud Controls Matrix. The framework provides guidance on which security controls should be implemented in container development.
    - Example controls
        - Access controls
        - Audit and accountability
        - Assessment, authorisation and monitoring
        - Identification and authentication
        - Risk assessment
        - System and communications protection
        - Configuration management
        - System and information integrity
        - System and services acquisition
        - Incident Response
- Vulnerability Tools
    - Grype
    - Dagda
    - Clair
    - Trivy

- Container Images
    - 20 images will be used for testing across each tool

        - The project will use the same images for each tool. The images range across multiple categories such as operating systems, messaging services and programming languages.

Aims of Project

- To investigate container security controls and analyse the effectiveness of vulnerability assessment tools on container images.

Objectives

1. Review previously published similar works
2. Establish the uses of containers in large-scale enterprise workloads
3. Research tools and policies that provide security to containers and infrastructure
4. Perform in-depth analysis of container security, including security controls and features
5. Plan, set up and configure container vulnerability tools
6. Test popular container images for OS and non-OS vulnerabilities
7. Provide in-depth analysis of the number of vulnerabilities found in container images for each tool used.
8. Evaluate the effectiveness of vulnerability tools
9. Propose a secure container policy that can be applied to safeguard the use of containers by enterprises.

Skills

Familiar Knowledge:

- Virtual Machines
- Networking
- Kali Linux

New Skills:

- Container Vulnerability Tools
- Working with container images

New skills acquired throughout the project will be done through thorough research of the tools and policies within container security.

Bibliography

AppDynamics, 2022. *Why DevOps is Important.* [Online]
Available at: https://www.appdynamics.com/topics/why-devops-is-important
[Accessed 7 March 2022].

AVI Networks, 2022. *Microservices And Containers.* [Online]
Available at: https://avinetworks.com/what-are-microservices-and-containers/
[Accessed 16 January 2022].

Barua, H., 2020. *Half of 4 Million Public Docker Hub Images Found to Have Critical Vulnerabilities.* [Online]
Available at: https://www.infoq.com/news/2020/12/dockerhub-image-vulnerabilities/
[Accessed 18 December 2021].

Bélair, M., Laniepce, S. & Menaud, J.-M., 2019. *Leveraging Kernel Security Mechanisms to Improve Container Security: A Survey,* Canterbury: Association for Computing Machinery.

Binnie, C., 2018. Auditing Docker Containers in a DevOps Environment. *ADMIN Real World AWS,* Issue 43, p. 100.

Brady, K., Moon, S., Nguyen, T. & Coffman, J., 2020. *Docker Container Security in Cloud Computing,* Las Vegas: IEEE.

Czikó, Z., 2019. *Management and Vulnerability Scanning of Docker Containers in an Embedded Enviroment,* Budapest: Budapest University of Technology and Economics.

Duncan, B., Lee, Y. & Olmsted, A., 2018. *Cloud Computing 2018 Proceedings of the Ninth International Conference on Cloud Computing, Grids, and Virtualization,* Barcelona: IARIA.

García, V. H., 2021. *NIST 800-53 compliance for containers and Kubernetes.* [Online]
Available at: https://sysdig.com/blog/nist-800-53-compliance/
[Accessed 31 January 2022].

Gartner, 2020. *Gartner Forecasts Strong Revenue Growth for Global Container Management Software and Services Through 2024.* [Online]
Available at: https://www.gartner.com/en/newsroom/press-releases/2020-06-25-gartner-forecasts-strong-revenue-growth-for-global-co
[Accessed 15 January 2022].

Github, 2022. *Trivy.* [Online]
Available at: https://github.com/aquasecurity/trivy
[Accessed 14 Febuary 2022].

Haymore, A., 2022. *10 real-world stories of how we've compromised CI/CD pipelines.* [Online]
Available at: https://research.nccgroup.com/2022/01/13/10-real-world-stories-of-how-weve-compromised-ci-cd-pipelines/
[Accessed 15 Febuary 2022].

Khandelwal, S., 2017. *It's 3 Billion! Yes, Every Single Yahoo Account Was Hacked In 2013 Data Breach.* [Online]
Available at: https://thehackernews.com/2017/10/yahoo-email-hacked.html
[Accessed 16 December 2021].

National Cyber Security Centre, 2021. *Defending software build pipelines from malicious attack.* [Online]
Available at: https://www.ncsc.gov.uk/blog-post/defending-software-build-pipelines-from-malicious-attack
[Accessed 20 January 2022].

National Institue of Standards and Technology, 2017. *Application Container Security Guide,* s.l.: U.S. Department of Commerce.

National Institute of Standards and Technology, 2017. *Application Container Security Guide,* s.l.: U.S. Department of Commerce.

NIST, 2020. *Security and Privacy Controls for Information Systems and Organizations.* [Online]
Available at: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf
[Accessed 9 March 2022].

North Carolina Department of Information Technology, 2022. *Assesment, Authorization and Monitoring Policy.* [Online]
Available at: https://it.nc.gov/media/1751/open
[Accessed 8 March 2022].

Peacock, J., 2022. *NIST SP 800-53 Control Families Explained.* [Online]
Available at: https://www.cybersaint.io/blog/nist-800-53-control-families
[Accessed 27 01 2022].

Red Hat, 2018. *What is CI/CD?.* [Online]
Available at: https://www.redhat.com/en/topics/devops/what-is-ci-cd
[Accessed 12 Febuary 2022].

Rice, L., 2021. *Vulnerability Scanning for Kubernetes Applications: Why and How.* [Online]
Available at: https://blog.aquasec.com/kubernetes-vulnerability-scanning
[Accessed 25 Febuary 2022].

Scroxton, A., 2021. *Low-complexity CVEs a growing concern.* [Online]
Available at: https://www.computerweekly.com/news/252496201/Low-complexity-CVEs-a-growing-concern
[Accessed 15 March 2022].

SecureCloudDB, 2021. *4 Major Breaches in 2020 and How They Could Have Been Mitigated (Part 1).* [Online]
Available at: https://www.secureclouddb.com/blog/4-major-breaches-in-2020-part-1-estee-lauder#:~:text=Half%20a%20Billion%20Records%20Exposed,was%20discovered%20in%20January%202020.
[Accessed 23 Febuary 2022].

Souppaya, M., Morello, J. & Scarfone, K., 2017. *NIST Special Publication 800-190,* Maryland: NIST.

Sultan, S., Ahmad, I. & Dimitriou, T., 2019. *Container Security: Issues, Challenges, and the Road Ahead,* s.l.: IEEE.

Sysdig, 2022. *CI/CD Security: Securing Your CI/CD Pipeline.* [Online]
Available at: https://sysdig.com/learn-cloud-native/container-security/cicd-pipeline/
[Accessed 21 Febuary 2022].

Trend Micro, 2019. *Docker Hub Repository Suffers Data Breach, 190,000 Users Potentially Affected.* [Online]
Available at: https://www.trendmicro.com/vinfo/es/security/news/cybercrime-and-digital-threats/docker-hub-repository-suffers-data-breach-190-000-users-potentially-affected
[Accessed 17 December 2021].

Trend Micro, 2021. *Top 10 AWS Security Misconfiguration.* [Online]
Available at: https://www.trendmicro.com/en_hk/devops/21/k/top-10-aws-security-misconfigurations.html
[Accessed 23 Febuary 2022].

VMware, 2021. *AC-6 LEAST PRIVILEGE.* [Online]
Available at: https://docs.pivotal.io/nist/ac/ac-6.html
[Accessed 10th January 2022].

VMWare, 2022. *AU-2 AUDIT EVENTS.* [Online]
Available at: https://docs.pivotal.io/nist/au/au-2.html
[Accessed 2 March 2022].

VMWare, 2022. *CM-3 CONFIGURATION CHANGE CONTROL.* [Online]
Available at: https://docs.pivotal.io/nist/cm/cm-3.html
[Accessed 3 March 2022].

Warrier, A., 2020. *Containers vs Virtual Machines (VMs).* [Online]
Available at: https://www.eginnovations.com/blog/containers-vs-vms/
[Accessed 18 January 2022].

Resources

Software required:

- Open-source container vulnerability tools
    - Vulnerability tools will be obtained through GitHub repositories
- Open-source Container Images
    - Container images needed for the project will be obtained through GitHub and Docker Hubs container image library
- VMware

- As the open-source vulnerability tools require a Linux operating system, VMware is needed to run the OS.

Report structure

1. Declarations
2. Acknowledgements
3. Abstract
4. Introduction
5. Literature Review
6. Uses of containers
   a. Cloud Networks
      i. Microservices
      ii. DevOps
      iii. Hybrid and multi-cloud environments
      iv. Application modernisation
   b. Enterprise workloads
7. Analysis of container security
   a. Controls
   b. Features
8. Vulnerability tools
   a. Tools
   b. Policies
9. Vulnerability tool setup
   a. Tool Configuration
   b. Docker Image Setup
10. Container image testing
    a. Grype
    b. Dagda
    c. Clair
    d. Trivy
11. Analysing the vulnerability found in container images
12. Evaluating the effectiveness of vulnerability tools
13. Evaluation, Conclusions & Recommendations
    a. Analysing the results
    b. Security Recommendations
       i. Secure container policy
    c. Evaluation of findings.
    d. Conclusion

Project Plan

Milestone/Time bounds	*Date*
Begin Research & Planning	w/c 15th November

Begin Literature Review	w/c 15th November
Plan Investigation Process	w/c 22nd November
Start Research on Container Uses, Tool and Policies	w/c 29th November
Submit Research & Planning Chapters	10th December
Begin Practical Work	24th January
Complete Practical Work	28t March
Plan Final Submission	25th April
Submit Final Project	5th May
Complete Viva	27Th May

Terms of Reference Review Form

This form is to be completed **by 2nd marker** after the TOR group review meeting and then sent to the student and the supervisor for signature. After the TOR group review, students should make any necessary changes to the TOR, online ethics and risk assessment forms (as required by your Supervisor and Second Marker), then submit the final approved TOR to Blackboard by the dates given in the module schedule whilst also completing the online ethics and risk assessment approval process.

Date of TOR Review:

Review Outcomes:

The topic is appropriate to the student's programme.	Yes
The project contains sufficient practical work using computing skills relevant to the programme.	Yes
An appropriate topic for the literature review has been identified.	Yes
An explanation of the contribution of the analysis/literature review to the project work has been given.	Yes

The TOR (select one):

• Accepted without changes.	
• Needs the changes listed below.	x
• Cannot be made satisfactory and a new topic is required.	

Ethics Draft PDF (select one):

• Has been reviewed and can be approved via the Ethics Online System.	x
• Requires revision before approval can be granted via the Ethics Online System.	
• Has been reviewed; the project should be referred to the Faculty Research Ethics Committee (FREC) via the Ethics Online System.	
• The project has already been referred to FREC via the Ethics Online System.	
• Has not yet been provided.	
• Other (please explain).	

Risk Assessment Draft PDF (select one):

• Has been reviewed and can be approved via the Ethics Online System.	
• Requires revision.	

• Is required but has not yet been provided.	
• Not required.	x

Changes required/identified issues:

Changes to proposed aim(s): No need to have 3 separate aims, combine them into 1. **Change "usability" to "effectiveness" of assessment tools**
Changes to proposed objectives: Objective 2 is vague, what do you mean by establishing… is it part of the practical work? Objective 6&7 are to investigate vulnerabilities of container images – if this is part of your work, need to reflect this in the aim.
Changes to deliverables:
Changes to structure and contents of project report:
Changes to project plan:
Resource issues:

Other comments: Please constraint the project in terms of feasibility. How many container images and assessment tools will you investigate?

Signatures: **Student** Jonathan Hill

Supervisor Rafay Ansari

Second Marker Xiaomin Chen