Abertay University

Kubernetes-Based Environmental Security Analysis:

an Evaluation of Techniques Employed to Secure Kubernetes Environments and Their Effectiveness.

Finlay Reid
BSc (Hons) Ethical Hacking 2021/22

Supervised by Natalie Coull (n.coull@abertay.ac.uk)

School of Design and Informatics
Abertay University

Table of Contents

Contents

Table of Figures

Table of Tables

Acknowledgements

I want to thank my supervisor Natalie Coull for her assistance throughout this project. The meetings almost every week helped guide me in writing this dissertation immensely, and I appreciate it. Furthermore, I wish to mention my family, particularly my mother, for being supportive during my four years at university.

Abstract

Kubernetes is container management software developed by Google, assisting in deploying and orchestrating container virtualisation technology. The ever increasing uptake of both containers and software that manages this technology has met concerns related to its security or rather lack off. Currently, Kubernetes is utilised by a host of essential organisations, and with good reason; the software has the ability to improve existing products or assist in the development life cycle. The software consists of several vital components that deal with different aspects of computing, and there exist certain relationships between components within Kubernetes environments. Configuration of the environment can be achieved through modifying the Kubernetes resource files. However, this is where security faults are born. Despite the progress made with the development of numerous practical Kubernetes security tools and the release of in-depth security material, the threat posed by exploited security issues remains high.

This paper documents the testing of widely used static analysis tools to examine the current state of Kubernetes security. Five of these static analysis tools were selected, with the resource usage and efficiency being recorded, assisted by the gnu utility. In addition, a Kubernetes hardening script was developed in the python programming language capable of performing various checks ranging from file permissions to admission controller tests.

The static analysis tool experiment demonstrated that the best overall tool currently available is Checkov. With its wide range of checks, the wealth of documentation available and readable output. At the same time, the development of the Kubernetes hardening script assisted in identifying areas of improvement for Kubernetes security tools. Furthermore, these practical tasks, including the research phase, helped examine the most pertinent issues Kubernetes environments face.

Keywords: Kubernetes, Kubernetes security, containers, containerisation, virtualisation, hardening script, static analysis

Abbreviations, Symbols and Notation

Abbreviations	**Meaning**
CPU	Central Processing Unit
API	Application Programming Interface
IP	Internet Protocol
HTML	Hyper Text Markup Language
PDF	Portable Document Format
CLI	Command Line Interface
CVE	Common Vulnerabilities and Exposures
VM	Virtual Machine
OS	Operating System
POC	Proof of Concept

Chapter 1 Introduction

1.1 Background

Since the previous decade, there has been a boom in companies migrating to virtualised environments. A shift brought about by the plethora of benefits promised to users and the numerous problems with traditional computing. However, a new form of virtualisation has seen significant growth due to reasons including high portability, a strong relationship with cloud technologies and the adoption of methodologies like Agile, Kanban and DevOps processes. Companies are making the dive into container-based virtualisation and are not regretting it. This technology is a lightweight alternative to virtual machines and, as previously mentioned, significantly increased the use of virtualisation by users, allowing software to be run without worrying about platform compatibility. Containers can be deployed with relative ease within the cloud, and several companies provide both cloud-based services and a container-based hosting solution. However effective containerisation was, there required some way to manage this hugely popular virtualisation technology, which brought about the invention of container management systems.

Kubernetes is a prevalent container orchestration system used by many leading companies and organisations, including Adidas, Nokia, Spotify, and the U.S. Department of Defense (DoD) (Case Studies, n.d.). Kubernetes employs its own terminology and architecture, enabling users to deploy containers using their chosen images. Benefits are abundant when using this software, as evidenced by the enormous reductions reported by Adidas for load time in their online businesses. There was a substantial decrease in half their load times, not to mention the increase in release frequency from initially every 4 to 6 weeks to 3 to 4 times a day(adidas Case Study, n.d.). Despite the substantial benefits Kubernetes has to offer, concerns have been raised over how secure this software is and what the developers at Kubernetes have in the pipeline to soothe these anxieties. Numerous studies have been conducted to back this up, including a survey of over 500 DevOps, engineering and security practitioners for the summer red hat 2021 report. Within the contents of these surveys, it is suggested that 94% of respondents stated they had experienced a security issue in their Kubernetes and container domains during the last 12 months (The State of Kubernetes Security, 2021).

A survey shown below in **Figure 1** conducted by the cloud-native computing foundation demonstrates organisations' continued adoption of Kubernetes. Kubernetes manages 69% of containers at the organisations surveyed. Furthermore, of the organisations that use the google container engine, 85% disclosed they also use a generic type of Kubernetes (E Hecht, 2018).

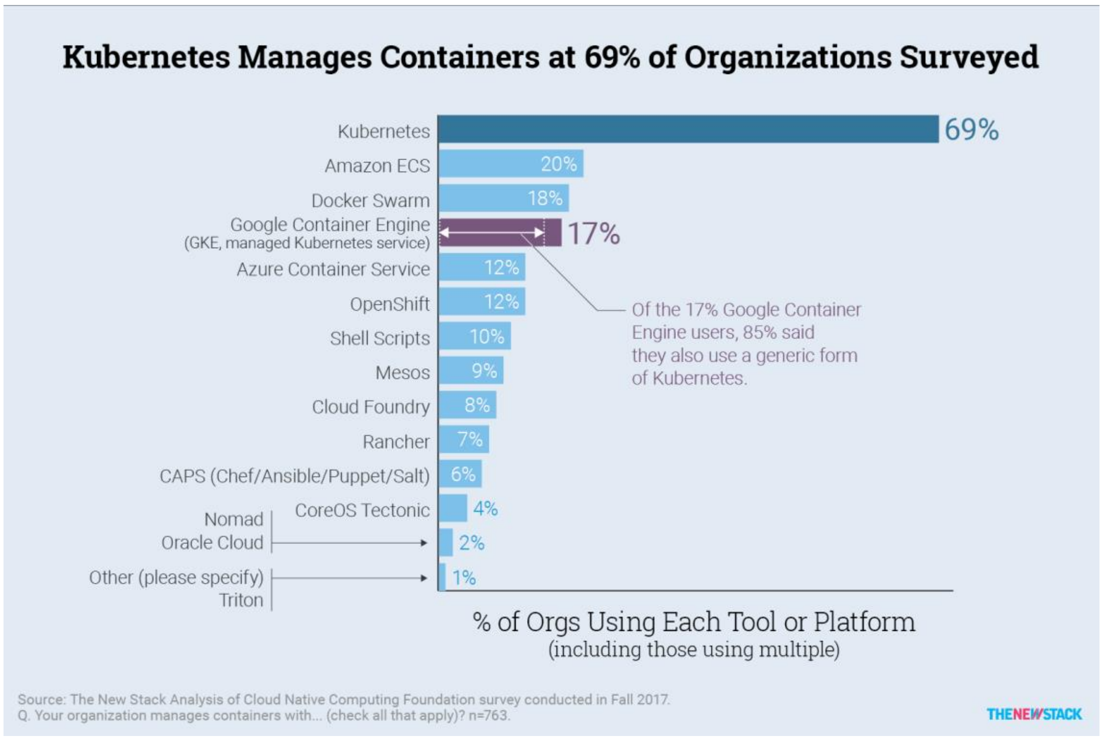


Figure 1 - Number of Orgs Using Each Tool or Platform

In order to ensure the Kubernetes environment remains secure as possible, it is necessary to take a defence-in-depth approach. There are numerous points of entry for an attempt at exploiting the Kubernetes environment. For instance, as Recent as 2018, users with ill intentions gained unauthorized access to Teslas Amazon Web Services resources by exploiting a vulnerability in a Kubernetes console (Team et al., 2018). Hardening orchestration systems such as Kubernetes require users to take the initiative to seek weaknesses found within proactively. The software has inbuilt mechanisms to protect against exploits, but these are not sufficient. Therefore, the onus is placed on the user to protect themselves when using this container orchestration software. As evidenced by a global survey in 2021 and presented in

Figure 2, security is the most challenging aspect of Kubernetes to overcome. At 54%, only data management and cross data centre support concerns come close (Vailshery, 2022).

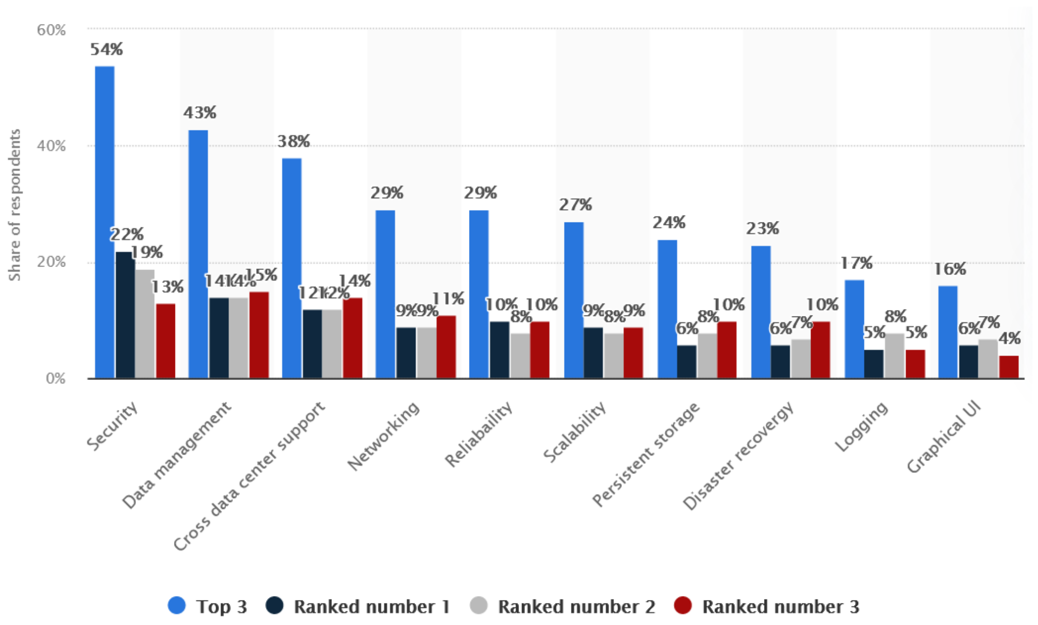


Figure 2 - Primary Challenges for Using Kubernetes in Organisations Worldwide 2021

In order to assist users in securing their Kubernetes environment, various tools have been developed—these range from remote testing services to static code analysers of configuration files. Overall there has been continual development of Kubernetes security tools, with new tools improving on old methods and covering different security niches. A healthy community of developers has ensured that the base implementation of Kubernetes security can be significantly improved by utilising their tools.

1.3 Research Question

What are the most significant issues pertaining to Kubernetes-based environmental security, and how can these be mitigated through testing existing static analysis tools and the development of a hardening script?

1.4 Aims and Objectives

Aim - The following project aims to complete a meticulous examination of Kubernetes-based environmental security, analyse different static analysis tools and develop a competent Kubernetes hardening script.

Objectives –

- Development of a Kubernetes hardening script that performs checks on several different aspects of a Kubernetes environment and presents the results in a readable format.
- Identify and investigate existing research related to Kubernetes best practices and the state of Kubernetes/container security.
- Conduct an experiment measuring the efficiency and resource usage of prevalent Kubernetes static analysis tools.

1.5 Structure

The second chapter of this paper examines literature pertinent to the project, including but not limited to Kubernetes security issues, Kubernetes tools and container malware. The methods and processes utilised in executing the project will be described in the third chapter of this paper. Including how the Kubernetes hardening script was developed and the static analysis tools tested. The following two chapters display and analyse the results garnered from the previous chapter, evaluating the developed artefact and comparing the final implementation against previous work. Finally, the paper is brought to a close with a discussion regarding future work and a conclusion of the paper's findings.

Chapter 2 Literature Review

2.1 Introduction

The following chapter investigates material relevant to the state of Kubernetes security and critically reviews the information found within peer-reviewed documents. Topics discussed include Kubernetes security tools and the attacks that test Kubernetes environments. The investigation is undertaken to determine the current knowledge of Kubernetes security, the tools available to security practitioners, and the methods in which Kubernetes/Containers are attacked. Knowledge gained from this will assist in developing the project's artefacts and the execution of the methodology. Firstly, using only credible sources, Kubernetes security best practices are discussed by examining previous research in the field.

2.2 Kubernetes Security Concerns and Issues

Numerous security concerns relate to the container orchestration system known as Kubernetes. As a result of these concerns, security practitioners have developed best practices and accompanying documentation required to improve the security of Kubernetes environments. Furthermore, due to the many components present within a Kubernetes system and the lack of research into Kubernetes security, users must implement as many of these best practices as possible (Lin et al., 2018). For instance, one example of a best practice involves users applying authentication and authorisation rules; this action can prevent would-be attackers from gaining unauthorised access to the cluster. In addition, these rules are typically disabling or enabling certain features within Kubernetes through the config files or can be implementing specific technologies that are more secure.

One example of this includes enabling the admission controllers. In Kubernetes environments, admission controllers are responsible for grabbing requests made to the Kubernetes API following authentication and authorisation but before they are executed or stored in etcd. Due to this, they remain one of the best methods for implementing security controls. Furthermore, it is possible to improve Kubernetes workloads through the use of admission controllers as built-in PodSecurityPolicy mounts the root file system as read-only and prevents the containers from running as the root user (Mytilinakis, P., 2020).

Checkov a tool discussed in 2.3 Kubernetes Tools/Scripts has many checks related to authentication and authorisation, as it is such a vital part of securing Kubernetes. For example, checkov preforms many checks related to authorisation including but not limited to ensuring the authorization-mode argument includes RBAC on all Kubernetes entities and that the anonymous-auth argument is set to false. The majority of these tools examine the relevant YAML files, such as the Kube-API file server, ensuring that the configuration is in line with recommended best practices such as setting resource limits and implementing read-only filesystem for containers where possible. Instead of manually checking the files to determine if admission controllers are enabled, the software will perform this on its own, telling the user the result and the process if it needs correcting. When in a Kubernetes based environment, authorisation correlates to the assessment of Kubernetes authenticated API requests based on the implemented rules that have been designated in the YAML files (Shamim et al., 2020). In order to determine if the request should be accepted or denied. Practices related to authorisation and their implementation can help reduce the risk of privilege escalation attacks, which is discussed further in 2.4 Kubernetes Attack Vectors/Surfaces. On the other hand, authentication refers to the validation of a user through external plugins, typically using client certificates, bearer tokens, or an authenticating proxy(Burns, B. and Tracey, C., 2018).

Tasks necessary to implement the practice of authentication and authorisation include disabling anonymous access to the Kubernetes server, as, by default, Kubernetes allows anonymous access to the Kubernetes API server. Failure to do this opens the cluster up to a range of attacks and fingerprinting. With a command such as curl and the knowledge of the master server and its port, it is possible to enumerate many directories relevant to the API server. It might not be a critical vulnerability to the system; however, it gives attackers a good base of knowledge to start their attack. Previous research shows that finding clusters set to accept anonymous connections are common(Shamim et al., 2020). An essential point of best practice includes mitigating the risk posed by user impersonation, as Kubernetes allows users to mimic others through impersonation headers. The feature functions by having the API request override the user info a request authenticates as. While in some circumstances, this has its uses, i.e. it is used by admins to debug the authorisation process(Shamim et al., 2020). Users with malicious intent can exploit this if the proper rules have not been configured. Ultimately, this can be hugely damaging to a Kubernetes environment. Again this can be related to the risk posed by privilege escalation; if practical limitations have been defined, malicious users do not have the option to use Kubernetes impersonation maliciously to obtain higher privileges.

Concerning the types of technology to use for authentication and authorisation to remain secure, security practitioners recommend using OpenID. This technology is an open standard decentralised authentication protocol allowing end-users to communicate with a relying party - A server that gives access to a software application. Within a Kubernetes environment, an OpenID connect token can identify the user; this token is a JSON Web Token (JWT) with well-known fields, such as a user's email, signed by the server. The authenticator uses the id token from the OAuth2 token response as a bearer token. Due to all the relevant data stored in the id token, the method is stateless. However, this method is not faultless due to the id's inability to be revoked, meaning a new token is required every few minutes, which can be inconvenient (D'Silva, D. and Ambawade, D.D., 2021).

2.3 Kubernetes Tools/Scripts

Several tools and scripts have been developed to help address the security issues and concerns with Kubernetes that will was discussed in 2.2 Kubernetes Security Concerns and Issues. These tools include Kubepatrol, Kubesec, Kube bench and Kube hunter(Rangta, M., 2022). The tools are designed to address different issues, and their effectiveness will be discussed here. Kubepatrol is an automated security auditing tool that aims to build on existing standard security tools (e.g. NMAP). It provides enhanced security for Kubernetes through the Kubernetes Python client, allowing the tool to perform checks on the configuration files produced by the container orchestrator. Implementing the python client ensures Kubepatrol can successfully harden Kubernetes environments, while its development has centred around Kubernetes best practices. It follows a list of security controls covering Network Security policies enforcement, pod security and port scanning. Unlike many of its rivals, the tool was developed with an emphasis on network and configuration scanning at its core; this is evident in its use of NMAP and network policy checks. As network scanning is an essential phase of hardening, due to benefits including ensuring the network is in good health and improving the visibility of the cluster, the Kubepatrol tool has a considerable advantage over many of its competitors.

Despite the many advantages of Kubepatrol, there remain several limitations to the tool and its design. Foremost, with its primary focus on auditing Kubernetes configuration files, it fails to have an aspect that performs permission checks on the actual files themselves in order to maintain the integrity of the file. Files should be writable by only the administrators on the system. Furthermore, the tool only provides suggestions on fixing some discovered security issues. Finally, it does not automate this process for the faults; instead, it relies on the user to carry this out. (Rangta, M., 2022).

A POC script was developed by a master's student at Dublin University to demonstrate how an integrated framework could be designed particularly for virtual infrastructure testing. Utilising several open-source tools such as Kube hunter and NMAP, the script provides users with a catalogue of potential exploits related to their Kubernetes implementation. The proof of concept script achieves this by scanning the pod's IP externally using the open-source tools and reviewing the output with the Mitre CVE database to provide the user with the related CVEs. The script can be broken down into three sections, including the running of the automated scans, processing of the results and formatting of the output. The final part of the code is responsible for creating a cve table, including all the vulnerabilities found in the previous pieces of code. As mentioned in the paper, the tool trivy had the most human-friendly output by utilising tables and colours. Therefore the POC script uses a similar method to display its results. However, the script in its current state is underdeveloped, requiring a significant number of hours to flesh out and add additional functionality (Fowley, T., 2021).

2.4 Kubernetes Attack Vectors/Surfaces

The specific types of exploits Kubernetes and container environments are hit with can include privilege escalation, denial of service, and sensitive information leakage. There is also a vast attack surface available to malicious users with all the components present within a Kubernetes system. Attack vectors include the node, API, container image and more. Following a traditional method when exploiting a system, information can be fingerprinted by using basic curl commands on the Kubernetes API server. It is possible as the API has a port exposed to the public allowing clients and other server modules to communicate with it. All of the attacks previously mentioned can be achieved through different methods, target many components and have varying levels of success, which will be thoroughly investigated in this section.

Privilege escalation is one of the most significant risks in a Kubernetes environment; when an attacker gains a hold in the system, they can leverage this to gain an increasing amount of access, ultimately leading to full access to the cluster. Privilege escalation is possible by exploiting improper following of symbolic links, Usage of the same volumes for different containers and more. One way to achieve this is through a technique that exploits a vulnerability when running a process as a mem cache user; the malicious user can gain unauthoried access to the container. Access to the container can be leveraged to obtain access to the entire cluster possibly. Every Kubernetes system container is typically made with a user-specified in the respective Docker file. However, this can be stopped by modifying the PodSpec file, which details how to run the containers or configuring the container to avoid running as the root user. Once Kubernetes starts up, it first looks to the docker file if nothing else was specified in PodSpec. The attack exploits this because sometimes the image can be run as root instead if certain conditions are met. For example, If mustRunAsNonRoot: true is specified, the container will refuse to start as uid 0, affecting availability. Mitigating the risk posed by this exploit involves users specifying 'runAsUser' directives in pods to control the uid a container runs as and downgrading Kubelets to v1.14.1 or v1.13.5(Artem et al., 2020).

Before a Kubernetes environment can function, it relies on container runtime technology to ensure the container can work on the host operating system. When using the container architecture, one of its tasks is pulling container images from repositories. The container management software and container runtime typically work hand in hand, whatever the software/type. Previous research into this aspect of the container ecosystem has concluded that container images used by many leading container runtimes are exceptionally vulnerable. Several researchers have come to this conclusion. For example, a study confirmed the fears held by security practitioners as more than 30% of docker (the Industry-leading container runtime) images were vulnerable to an array of differing attacks(Gummaraju et al., 2015).

Docker hub provides users with an extensive repository of uncontrolled images that anyone using the service can upload. It is currently the official repository of docker images and has the most significant users, with 650,000 registered to the resource.

Images are updated and maintained only by the users who created the repository. This means updates are few and far between, ensuring the images have many vulnerabilities. As there can be a significant amount of time between updates, possibly even months, and if an image is using outdated libraries, it has the chance of compromising the container system or the host operating system(Gummaraju et al., 2015).

One of the main issues explored by security partitioners within studies is the risk of a container breakout and the effects on the host operating system. Utilising kernel-level isolation in combination with kernel security mechanisms such as namespaces and cgroups, containers have the ability, in theory, to provide complete isolation. However, this is not always the case as the container isolation mechanisms can be bypassed, as evidenced by the study undertaken by researchers at Purdue University (Reeves et al., 2021).

2.5 Container Malware

Malware designed specifically around exploiting the container architecture is relatively new so there are few papers available, however, research can be found in a number of blogs and web pages. In particular, the malware that targets window containers named siloscape has had documentation including in-depth analysis provided by Palo alto networks (Prizmant, 2022). Palo alto's analysis of the container malware is the sole in-depth documentation as the company did not release the malware binary to other researchers. There are also several variations of container malware including a ransomware-type known as DarkRadiation which uses technologies such as wget, curl, sshpass. Using a telegram bot for communication the malware attempts to gain full persistence through a service called "griphon, then it starts the encrypting process while utilising an obfuscation program called node bash obfuscate. Taken at face value the obfuscation appears to be complex but it can be easily circumvented by changing the eval command with echo, this gives the output stdout without executing it. A case could be made that there is a slight bias in the blog due to sentinels one push of its own software throughout the analysis(SentinelOne. 2022).

2.6 Kubernetes network security issues/exploits

Just like a traditional computer environment, the Kubernetes cluster is susceptible to attacks over its network if it has not been sufficiently hardened. However, papers researching the security aspect of Kubernetes network components are scarce. The networking component of a Kubernetes cluster relies on networking addons such as Calcio , weave etc. Facilitating for the clusters containers to communicate across different nodes. When using a CNI plugin such as weave, each pod has its own unique bridge interface that connects all the local containers. Both CNI plugins weave and Calcio utilise iptables to apply network policies (Nam et al., 2020).

2.7 Summary

The research contained within the literature review chapter has explored tools security professionals use to improve security in their environment. The components these tools aim to secure, and the possible areas of attack for malicious users have been examined. Furthermore, concerns related to exploits and malware have been described. The aforementioned research has provided recognition of what are the most pertinent threats to Kubernetes security and, to a certain extent, how these issues could be mitigated through security tools, which this project builds on. From the research carried out in this chapter, the key issues in Kubernetes security are the limited functionality of tools aiming to secure clusters and the vast attack surface available to malicious users.

Chapter 3 Methodology

3.1 Overview

The methodology chapter discusses the process when undertaking the project. Explaining each of the primary stages in detail including the research, development and design. The research section compromises of analysing papers detailed in the previous section and research is undertaken to discover the most widely used Kubernetes security tools. Development consists of creating a Kubernetes hardening script that tackles the issues described in 2.4 Kubernetes Attack Vectors/Surfaces and improves on existing Kubernetes security tools. Finally, static analysis tools are tested and the testing methods are described. The steps undertaken in the methodology, in chronological order are:

- Research of Kubernetes set up, static analysis tools and script information.
- Set Up of testing environment.
- Installation of Kubernetes static analysis tools.
- Testing of Kubernetes static analysis tools
- Development of kubernetes hardening script
- Documentation of experiment and script development

3.2 Research

The research section of the project involved compiling several current Kubernetes security auditing tools. Ranging from basic static analysis tools that perform checks on the config files to full blown pen testing applications. Furthermore, the investigation into infrastructure set up was undertaken and the appropriate material related to Kubernetes/Container security including exploits, mitigations and resources was investigated. The stage also consisted of investigating previous academic research, namely papers and blogs that contained important information that could assist In the development section of the project. Papers were discovered using google scholar and other important information was found from security partitioners GitHub's. Sub topics investigated related to Kubernetes security included learning resources, CNI plugins, exploits, mitigations, container malware and others.

This phase of the methodology was essential as it enabled for the identification of the required steps necessary to develop the Kubernetes hardening script including deciding on the technologies and methods. Past experience with languages such as python and other technologies ensured research was primarily focused on Kubernetes.

3.3 Development Stage

The following section will first outline the setup of the locally provisioned Kubernetes cluster, and this infrastructure will facilitate the testing and implementation of the Kubernetes hardening script. Then, the specifics of the python script will be thoroughly discussed, detailing the design decisions made during the process based on the knowledge gained from literature in the previous section. Furthermore, the technologies and chosen methodology will be considered, including why and how.

This decision will enable the project to achieve its aim and answer the research questions.

3.3.1 Infrastructure Design

One locally provisioned cluster was created to test the Kubernetes hardening script. It was a clean installation of a Kubernetes cluster using hypervisor-based virtualization. It enabled the hardening script to be demonstrated on a semi secure configuration. In addition, this allowed the demonstration of the effectiveness of using the script and adhering to the appropriate Kubernetes practices recommended by security practitioners.

One machine was used during the configuration, testing, implementation, and research of the project. This Lenovo ThinkPad x1 carbon was equipped with hardware including an 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz and 16.0GB ram. The local operating system used on this machine throughout the project was the latest version of Windows 11.

3.3.1.2 Virtual Machine Configuration

The hypervisor software used throughout the project was called virtual box, a type 2 hypervisor service for the x86 architecture developed by the Oracle Corporation. The preferred choice when it comes to free Windows-based hypervisors. This enabled the creation of a Kubernetes cluster. The instance of the Kubernetes cluster was a clean installation, this required two separate virtual machines. These machines ran the latest versions of ubuntu 64-bit distributions of Linux running version 20.04.2.0.

These machines were named appropriately to avoid confusion, with one taking the role of the master node and the other worker node. This relationship is present within all Kubernetes clusters and is an important distinction when setting up the environment. One of the Virtual machines was titled ‘k8-Master’, and the other was called ‘K8-Worker’, which can be seen below in **Figure 3**. Other virtual machines could have been configured to add additional nodes, but for the sake of the project, a single worker and master node are sufficient. The virtual machines ran the default configurations apart from a few additions related to the network and base memory. Two network adapters were utilized, including a bridged adapter that would facilitate an SSH connection and a created NAT network that forms the cluster's IP addresses. Additionally, memory was added to ensure the virtual machine could cope with the running of the Kubernetes environment.

k8-Master (Reset Master)
Powered Off
K8-Worker (Reset Worker)
Powered Off

Figure 3 - VM Naming and Showcase

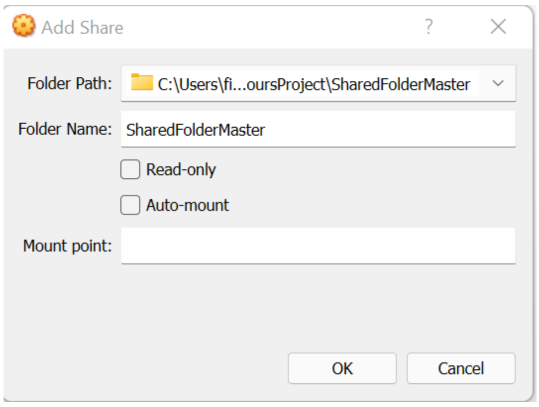
The process for implementing and configuring a Kubernetes cluster was taken from several online tutorials and the official Kubernetes documentation (Towett, 2022). The primary components used on the VM's are presented below in **Table 1**.

Components	Amount
CPU	1
Memory	2 GB
Network Adapters	2(Bridged/NAT)
Operating System	Linux Ubuntu Server 20.04

Table 1 - VM Components of the Two Machines

3.3.1.2.1 Shared Folder Setup

In order to transfer files between the host machine and the master node a shared folder was utilised. This would ensure the development process of the Kubernetes hardening script would be straightforward as the python script could be transferred back and forth. Virtual boxes in built shared folders functionality for its virtual machines was used, this was done through the installation of VirtualBox Guest Additions – a package consisting of device drivers and system applications including the addition of the shared folder functionality (Chapter�4.�Guest Additions, n.d.). The folder shown below in **Figure 4** is the location on the host machine's drive that is mounted.



Add Share

Folder Path: C:\Users\fi...oursProject\SharedFolderMaster

Folder Name: SharedFolderMaster

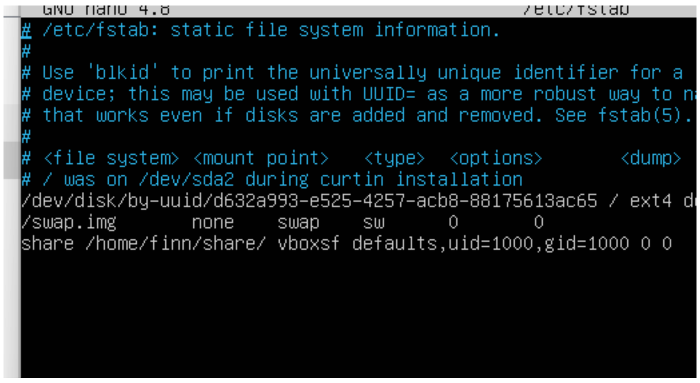
☐ Read-only

☐ Auto-mount

Mount point:

Figure 4 - Adding Shared Folder Config

In order to ensure the shared folder is automatically mounted on boot the /etc/fstab file is configured. The following data was inputted and saved as shown below in **Figure 5**.



```
  GNU nano 4.8                                        /etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to n
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>
# / was on /dev/sda2 during curtin installation
/dev/disk/by-uuid/d632a993-e525-4257-acb8-88175613ac65 / ext4 d
/swap.img       none    swap    sw      0       0
share /home/finn/share/ vboxsf defaults,uid=1000,gid=1000 0 0
```

Figure 5 - Shared Folder Mounted on Boot Set Up

3.3.2 Kubernetes Security Tools

This section introduces several static analysis tools to select the most suitable one for the Kubernetes hardening script. Advantages and disadvantages are considered, and with the specific checks each tool performs. Static security scanning enables security partitioners to perform checks on configuration files at the start of the development life cycle. This type of scanning ensures security standards are adhered to before the product is implemented or released. In the context of Kubernetes, static analysis tools inspect the YAML files for any security misconfigurations. That is why static analysis tools and YAML manifest linters are essential to check the security issues early in the manifest.

The static analysis tools investigated in this section include Checkov, Kubelinter, Kubeaudit, Kubescore and Kubesec. Following the literature review, it was determined that these tools were the most relevant based on the research carried out in that phase. In order to test these tools to determine what would be the most suitable for the hardening script, the GNU time utility was employed to measure the resource usage and time taken for each of the tools, which was accomplished through scanning the Kubernetes API YAML file using each of the security utilities. Furthermore, subjective aspects of the tools are discussed, including the readability of the output and the straightforwardness of the set-up process.

3.3.3 Kubernetes Hardening Script

3.3.3.1 Aim & Requirements

This script aims to successfully diagnose and implement fixes to Kubernetes environments based on security best practices. Furthermore, it would aim to include features that other tools lacked and improve on existing aspects of its predecessors. The script's primary requirement included easy-to-read results, as this aspect of security auditing tools is undervalued compared with the main functionality.

3.3.3.2 Programming Language & Modules & IDE

The Kubernetes hardening script will be developed in the programming language Python 3. Python is an extremely powerful high-level language that can be used for many purposes; it has numerous external modules that enable developers to create efficient programs and save vast amounts of time. In addition, specific tasks the script must complete are typically assisted by these external libraries created by the active developer community that the language possesses.

A modern IDE named PyCharm will be utilised to ensure the development process runs smoothly. This IDE is developed by the Czech company JetBrains, using the latest version on windows. In addition, features such as coding assistance, navigation and the debugger will be used to assist in the development of the script (PyCharm: the Python IDE for Professional Developers by JetBrains, n.d.).

The main python libraries necessary for the creation of the script will be described below; these assist in different aspects of the tool but are ultimately essential to make the script function as intended.

The python-Nmap library imports the Nmap port scanner permitting developers to run Nmap scans against desired targets. First released in 2010, the library's most recent version is the 0.7.1 release which will be utilised in this script. Python-Nmap allows system administrators to include automatic scanning tasks and reports (python-nmap, 2021).

Kubernetes has an official library that complements its orchestration system allowing developers to run commands related to Kubernetes but in the python language. Version 23.3.0, the latest release, is used in this project (Client Libraries, 2021).

The subprocess library allows developers to create new processes externally within a python program. Replacing the older OS module typically used for running commands from python programs, the subprocess module allows users to read the input/output. Version 3.10.4 is the most recent version which is used in the script (subprocess — Subprocess management — Python 3.10.4 documentation, n.d.).

Th termcolour module enables developers to change the colour of the text in the terminal. First released in 2008, the script utilises the newest version, 1.1.0. Both modules related to each of the file types, YAML and JSON, will be used as these files will be required in the script to parse and then pull information from the data (termcolor, n.d.).

3.3.3.3 Kubernetes Hardening Script Design & Process

The approach to developing the Kubernetes hardening script was iterative, requiring many prototypes throughout development; typically, a single aspect of the application was focused on. Iterative prototyping ensured continual improvements to the individual parts of the application resulting in the final product. The first step of development involved creating a program structure that would provide the base for the pseudocode. The structure that was opted for involved organising the files by Kubernetes components; for example, one file would primarily focus on the worker node, where all the checks, diagnoses and mitigations would occur for that particular Kubernetes component. Within the files, there exists a particular structure as well. At its core, there is a section that performs checks and another responsible for mitigating the faults found in the checks. After determining the most suitable structure, the next step in the development process was creating pseudocode that would assist in developing the actual script. The pseudocode was drawn up using a pen and paper, which was done roughly and required little time. The actual layout of the hardening script consisted of six files, including one for each of the primary Kubernetes components, one for the presentation of results, and the final one was responsible for the networking aspect of a Kubernetes cluster. This can be seen clearly in the diagram below in **Figure 6**. This design was chosen because it is the most logical, as several of the hardening guides researched in the literature review opted for this approach to Kubernetes hardening. Initially, another program structure was considered; this involved having a file for each check class. For example, one file would be responsible for scanning the files and ensuring they had the correct permissions, a permission checking file.

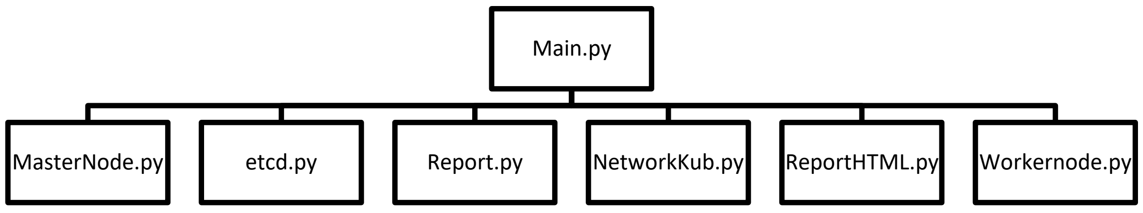


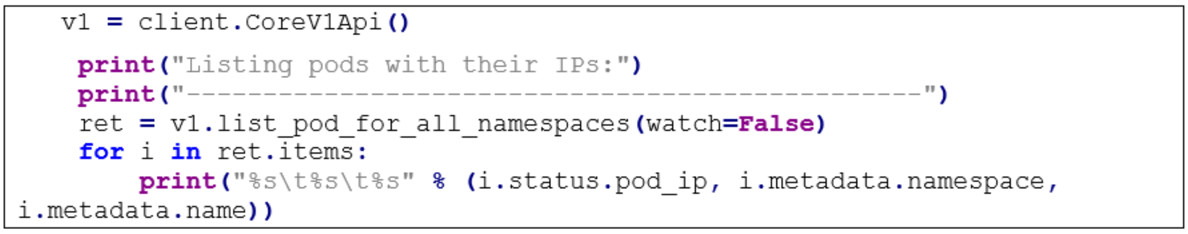
Figure 6 - Kubernetes Hardening Script Directory Structure

3.3.3.4 Main.py

The first task in developing the Kubernetes hardening script involved creating a file that would form the program's base. Ultimately, it was decided that this piece of the program would be responsible for pulling in all the other files, running them and giving the user relevant information about their Kubernetes environment. Therefore, all files responsible for performing checks on each Kubernetes component are loaded into the file. Declaration of files are contained within the main function, and when the main.py is executed, it runs all the component files sequentially. The full Main.py code can be seen in **Appendices – Main.py.**

Another aspect of the main.py included creating a section of code that provides the user with information relating to their Kubernetes implementation. Information such as this can be helpful to the user and confirms that the cluster is working as intended. Other Kubernetes security tools researched in the literature review apart from Kubescore failed to include anything similar to this feature. The official Kubernetes python library was determined to have the necessary features to enable this section to be programmed. In order to make use of this resource, the module first had to be downloaded on the testing environment and within the IDE, which was made straightforward through Pycharms python packages window. This process was repeated when installing the remaining modules.

In the Kubernetes environment, package installer pip was used to install all of the modules on the Kubernetes cluster. An instance of CoreV1Api Class from the kubernetes python module is stored in a variable enabling list_pod_for_all_namespaces() method to be called, giving the script access to pods on all namespaces present within the Kubernetes cluster. Finally, the stored pods are parsed, with relevant information such as the pod IP address and namespace printed to the user, as shown in **Figure 7**.



```
    v1 = client.CoreV1Api()

     print("Listing pods with their IPs:")
     print("-----------------------------------------------")
     ret = v1.list_pod_for_all_namespaces(watch=False)
     for i in ret.items:
         print("%s\t%s\t%s" % (i.status.pod_ip, i.metadata.namespace,
i.metadata.name))
```

Figure 7 - Print Pod IP Addresses Code

3.3.3.5 MasterNode.py

The MasterNode.py file ensures that the master node on the Kubernetes cluster has the correct security configurations implemented. Several checks are performed on the master node's configuration files. Including permission and file ownership of the files themselves and scanning the contents of the files, confirming they adhere to best practices. One instance of the checks is determining whether the appropriate resource limits are implemented, such as CPU and memory. If not, the user is notified of how to remediate this issue. From the reviewed literature, it was apparent that the current crop of Kubernetes hardening tools failed to include measures that ensured the config files themselves had the appropriate permissions. File ownership and permissions are crucial to maintaining the integrity of the config files, as they are responsible for altering the behaviour of Kubernetes services. The full MasterNode.py code can be seen in **Appendices – MasterNode.py.**

In order to successfully determine if the correct permissions and file access are set, the code shown below in **Figure 8** was developed. The snippet code is utilised throughout the file to check if the right permissions are in place. It is slightly altered when performing checks related to file ownership. This developed code functions by looping through a list of Kubernetes YAML files while utilising the subprocess library to execute the stat command directed at the file path of the configuration file. Utilising the -c and a% switches with Stat enables Linux users to report solely on permissions of the selected file. Output from the command is saved into a variable through subprocesses stdout and converted into an integer data type. This conversion permits a condition statement to compare the retuned output and the approximate level of permission that the file should have, i.e. 644. Should the file employ the correct permissions, i.e. greater than 644, the file is designated as a pass. The file is designated as a fail if the output fails to meet these conditions. Furthermore, if the check fails, it is rectified through the subprocess library and the chmod command to implement appropriate permissions.

```
    for cmd in cmds:
        PodPermissions = subprocess.run(["stat", "-c", "%a", cmd],
stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                                        text=True)
        output = (int(PodPermissions.stdout))
        test = output
        print(cmd)

        if test >= 644:
            pas = "pass"
            print(colored(pas, 'green'))

        elif test <= 600:
            pas = "fail"
            print(colored(pas, 'red'))
            for cmd in cmds:
                PodPermissions = subprocess.run(["chmod", "644", cmd],
stdout=subprocess.PIPE,
                                                stderr=subprocess.PIPE,
text=True)
```

Figure 8 - Permission Check Code On All Files

The following code presented in **Figure 9** of the master node file is the script's core. It involves checking the contents of the configuration files and is essential. Considering the results of testing and analysing the static analysis tools in the previous section, it was concluded that Checkov was the most suitable for the task at hand. Therefore, the subprocess library is employed, allowing a Checkov scan to be executed with the required switches.

The first line of code creates a file and saves the file object as a variable. The saved variable is then used as the output in the subprocess command. Next, a checkov report on a Kubernetes configuration file is saved in JSON format. Utilising the JSON Load function of the JSON library assisted in loading the file into a dictionary, this data is then parsed, and relevant data is pulled from it. Data scraped from the .json checkov report included the number of failed and passed checks specific to that Kubernetes configuration file. Again this code is used for all four master node YAML files, including the Kube-API server, Kube-controller-manager, etcd and Kube-scheduler.

```
    with open('kube-apiserver.json', "w") as outfile:
        test = subprocess.run(["checkov", "--compact", "--quiet", "-o",
"json", "-f", "/etc/kubernetes/manifests/kube-apiserver.yaml"],
stdout=outfile, stderr=subprocess.PIPE,
                                    text=True)

    with open('kube-apiserver.json', 'r') as json_file:
        json_load = json.load(json_file)


    passed = (json_load['summary']['passed'])
    failed = (json_load['summary']['failed'])

    print("Passed:")
    print(colored(passed, 'green'))
    print("Failed:")
    print(colored(failed, 'red'))
    print(colored("Full details and remadations can be found in kube-
apiserver.json", 'yellow'))
```

Figure 9 - Execution of Checkov Scan, Load of File and Parsing of Data

3.3.3.6 Nmap.py

Having investigated the current Kubernetes hardening tools used by users, looking at what they provide in ways of network security, it was concluded that there was a real lack of security measures related to Kubernetes networking. This absence of proper network security testing influenced the decision to include a file which singularly ran Nmap scans against the host's machine and Kubernetes services. Results from this would assist a user in discovering any unnecessary open ports and other important information for testing. The presented code in **Figure 10** functions by loading in the Kubernetes cluster configuration facilitated by the official Kubernetes Python library. Having imported the Kubernetes cluster information, the client package is utilised with the AppsV1Api class, granting access to the read_namespaced_service method. The data is stored into a variable and used to access the cluster IP address, which is printed to the user to confirm it is the correct Kubernetes service that was tested. The full Nmap.py code can be seen in **Appendices – Nmap.py.**

In the next part of the development, the socket module is employed to retrieve the machine's IP address. Possessing this data enables running a Nmap scan against the host's machine and its Kubernetes services.

After gathering the required data, the IP address is included in the opening Nmap scan. The scan function enables the script to run a basic scan against an IP address of choosing, storing the resulting data into a variable. Then, the resulting data is iterated with critical information, including open ports printed to the command-line interface.

```
config.load_kube_config()
    api = client.CoreV1Api()
    service = api.read_namespaced_service(name="kubernetes",
namespace="default")
    print("cluster ip:" + service.spec.cluster_ip)
    fin = service.spec.cluster_ip
    hostname = socket.gethostname()
    IPADDR = socket.gethostbyname(hostname)
    nm = nmap.PortScanner()
    scan_range = nm.scan(hosts=IPADDR)
    nm.all_hosts()
    for host in nm.all_hosts():
```

Figure 10- Code for Running Nmap Scan Against User IP Addresses

3.3.3.7 Report.py

The knowledge gathered in the literature review related to the current crop of Kubernetes security tools assisted in identifying the best methods of output. There was little time, and effort used when implementing the output functionality of one of some of these tools, which this tool aims to rectify by providing three forms of output. The python package report lab is utilised to create a dynamic PDF file that includes the results of a scan. Its design comprises a summary page and pages for each individual file (Report Lab User Guide, 2020). The layout of the summary page includes three frames and information such as the total sum of marks and the results in a graph format. This histogram is created dynamically in the ReportHTML.py file using the matplotlib python library (Matplotlib — Visualization with Python, n.d.). The following file pages all adhere to the identical format and contents with a single frame. The full Report.py code can be seen in **Appendices – Report.py** and the layout of the PDF pages can be viewed in **Figures 18 and 19.**

3.3.3.7.1 Summary Page

The same method utilised in the master node file to parse the JSON file and retrieve relevant information is again included in this file. The total is calculated and displayed in frame one. This data is colour-coded for readability, achieved through report labs paragraph flowable - an abstract base class for things to be drawn, and an instance knows its size and draws in its coordinate system (Report Lab User Guide, 2020).

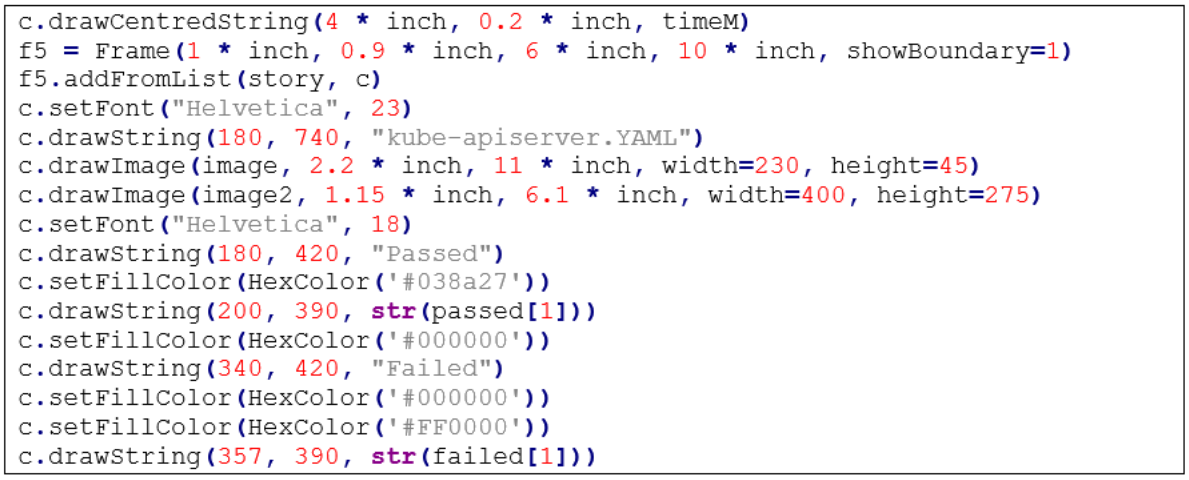
The first section of code shown in **Figure 11** is responsible for printing the data with the predefined styles implemented at the top of the file. Next, the time module is incorporated to give a unique name for the document. The name of the pdf report is always unique as its probable users' Kubernetes environments will change or be altered, ensuring the document is not overwritten; therefore, comparisons can be made.

```
story.append(Paragraph("The Cluster Score is:", style))
story.append(Paragraph("Passed:", stylePassedH))
story.append(Paragraph(str(passedT), stylePassed))
story.append(Paragraph("Failed:", styleFailedH))
story.append(Paragraph(str(failedT), styleFailed))
story11.append(Image('Summary.png',  width=330, height=305))
t = time()
t2 = str(t)
c = Canvas('PDF_Report' + t2 + '.pdf')
f = Frame(1 * inch, 9.4 * inch, 6 * inch, 1.5 * inch, showBoundary=1)
f.addFromList(story, c)
f.addFromList(story3, c)
f.addFromList(story4, c)
```

Figure 11 - PDF Document Summary Page Creation Code

3.3.3.7.2 Specific File Pages

The three file pages adhere to an almost identical format, but the data is substituted for that particular file. As previously mentioned, the pages consist of one large frame that encompasses almost the full paper, which is achieved through the use of the report labs frame object. The actual text contents on the page include the pass and fail totals, the title and the current data. This information is printed on the document through report labs methods drawCentredString. Furthermore, the draw image method is used to print two images per file, one is the pie chart, and the other is the Kubernetes logo. This shown below in **Figure 12**.



```
c.drawCentredString(4 * inch, 0.2 * inch, timeM)
f5 = Frame(1 * inch, 0.9 * inch, 6 * inch, 10 * inch, showBoundary=1)
f5.addFromList(story, c)
c.setFont("Helvetica", 23)
c.drawString(180, 740, "kube-apiserver.YAML")
c.drawImage(image, 2.2 * inch, 11 * inch, width=230, height=45)
c.drawImage(image2, 1.15 * inch, 6.1 * inch, width=400, height=275)
c.setFont("Helvetica", 18)
c.drawString(180, 420, "Passed")
c.setFillColor(HexColor('#038a27'))
c.drawString(200, 390, str(passed[1]))
c.setFillColor(HexColor('#000000'))
c.drawString(340, 420, "Failed")
c.setFillColor(HexColor('#000000'))
c.setFillColor(HexColor('#FF0000'))
c.drawString(357, 390, str(failed[1]))
```

Figure 12 - File Pages Creation and Design Creation Code

3.3.3.8 ReportHTML.py

The report HTML file involves loading the JSON file containing the results of a checkov scan, the method used to accomplish this was described previously in the Master Node file. The implementation differs from previous files regarding its need to parse for the class of the faults discovered. The code shown in **Figure 13** was developed to loop through failed checks. Within that, key-value pairs are iterated through also. Condition statements are utilised to count the occurrences of particular classes of checks. The collection of this data ensures that various graphs can be created. The full ReportHTML.py code is presented in **Appendices – ReportHTML.py.**

```
for index in range(len(test)):
       for key, value in test[index].items():
               # print(value)
               # print(test)
               if key == "check_class":
                   # y = [value]
                   # print(y)
                   counterTotal = counterTotal + 1

                   if "Capabilities" in value:
                       counterCapa = counterCapa + 1

                   elif "RootContainers" in value:
                       counterRootC = counterRootC + 1

                   elif "Limits" in value:
                       counterLimits = counterLimits + 1

                   elif "Service" in value:
                       counterSeriveA = counterSeriveA + 1

                   elif "Image" in value:
                       counterImage = counterImage + 1
```

Figure 13 - Check Class Counter Code

Knowing this data means a pie chart can be created with the help of the matplotlib library, and the counter information is inputted. Then, appropriate titles, labels and other adjustments are applied before saving the figure as a PNG file, as shown in **Figure 14**.

```
    Misc = counterCapa + counterRootC + counterSeriveA + counterImage +
counterLimits
    MiscCalc = counterTotal - Misc
    CounterT = [counterCapa, counterRootC, counterSeriveA, counterImage,
MiscCalc]
    labels = 'Capabilities', 'RootContainer', 'Limits', 'Service', 'Image'

    fig1, ax1 = plt.subplots()
    ax1.pie(CounterT, labels=labels, autopct='%1.1f%%',
            shadow=True, startangle=90)
    ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a
circle.
    ax1.set_title("Distribution of Failed Checks in kube-scheduler.yaml")
    plt.show()
    plt.savefig("kube-scheduler.png")
```

Figure 14 - Creation of Graphs Using Matplotlib Code

3.3.3.9 Other files

The remaining files within the script utilise much of the functionality described already. For instance, the file responsible for performing checks on the worker's node configuration files borrows code from the master node; however, where applicable, code is substituted and tailored so the file tests the worker node. The core functionality of the script is located within the first files, as described in depth above at **3.3.3.5 MasterNode.py.**

Chapter 4 Results

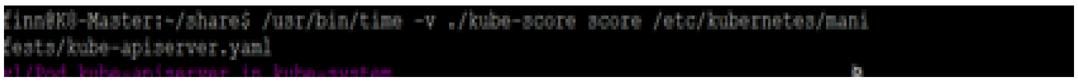
The following chapter will provide a presentation and elucidation of results gained from testing the static analysis tools. Individual tools were measured five times to ensure the results were sound when testing the static analysis tools. Furthermore, the final implementation of the Kubernetes hardening script, having developed the artefact following the methods shown in the methodology, will be described. Success in developing the Kubernetes hardening script is determined by the improvement of pre-existing tools and how the checks and features compared to the tool developed in this paper. However, some of these successes are subjective, for example, the tool's output.

4.1 Static Analysis Tools Testing

The testing of the Static Analysis Tools occurred pre-development of the script as it was crucial to determine what tools provided the most coverage and were the most efficient. Determining the tools that would be tested through this experiment was based on the research undertaken in the literature review. Five tools of varying repute and ability were selected. Software was installed through similar methods and set up in the Kubernetes environment.

The utility that measured these tools' performance is the gnu time utility available on Linux systems. A tool which enables users to record the resource usage utilised by another program and calculates times involving the program, including system time - the amount of CPU time spent in the kernel within the process. All command and outputs of the tools can be seen in **Appendices – Static Analysis Tools Commands & Output.**

The command shown below in **Figure 15** was the format of statement used on the CLI throughout testing, with varying tools substituted depending on the tool currently tested. Furthermore, the commands were altered slightly depending on the switches specific to that tool. Tools were tested five times to ensure an outlying result did not influence the final decision. The testing of the tools was also completed in one session in the same environment, utilising the same hardware and software. However, no results are perfect and immune external factors.



```
finn@K8-Master:~/share$ /usr/bin/time -v ./kube-score score /etc/kubernetes/mani
fests/kube-apiserver.yaml
v1/Pod kube-apiserver in kube-system
```

Figure 15 - Format of Command Used to Measure Resource Usage of Tools

Looking at the results shown in **Table 2,** the Checkov tool developed in python recorded a high of 12.21 seconds in user time and a low of 2.33. Its CPU usage varied throughout testing but mostly remained around the 70 mark. Across the board, the results remained similar apart from the outlier recorded at the start of the experiment. Kubeaudit was the third tool measured; it shared similarities with the results recorded when testing Kubelinter. Its system times were consistent in every test at 0.01, and its highest recorded user time topped out at 0.04. The penultimate tool measured was Kubesec; the results were consistent for every metric with no outliers. The only notable aspect of this tool results was the extremely low CPU usage throughout testing, with the highest at 16 and lowest at 8.

Tool	Language	Peak memory usage(kbytes)	User Time(seconds)	Percentage of cpu(%)	System time(seconds)	Elapsed time(user time)(seconds)
Chekov	Python	166372	5.56	51	0.74	12.21
		166820	1.77	87	0.26	02.33
		166752	1.88	71	0.20	02.92
		166544	1.90	65	0.18	03.16
		166764	1.89	72	0.23	02.94
Kubelinter	GO	33152	0.02	75	0.00	00.03
		32972	0.03	70	0.00	00.04
		33368	0.02	76	0.00	00.02
		33136	0.02	88	0.00	00.02
		33224	0.02	74	0.00	00.03
Kube audit	GO	14996	0.01	86	0.01	00.03
		14984	0.01	70	0.01	00.02
		14992	0.02	75	0.01	00.04
		15000	0.00	80	0.01	00.03
Kubesec	GO	19364	0.00	8	0.02	00.38

		19296	0.01	12	0.02	00.24
		19580	0.00	11	0.02	00.23
		19600	0.02	16	0.02	00.29
		19284	0.02	16	0.02	00.31
Kube score	GO	8916	0.01	70	0.01	00.03
		8912	0.01	78	0.01	00.04
		8984	0.03	80	0.00	00.04
		8912	0.01	73	0.02	00.04
		8968	0.02	78	0.01	00.03

Table 2 - Static Analysis Tools Experiment Results

4.2 Completed Kubernetes Hardening Script

The final implementation of the Kubernetes hardening script comprises seven individual files that each provide varying features and advantages to the developed artefact. Functionality is combined into one file that enables users to execute the script from the CLI. The tool aims to assist users and implement as many security best practices as possible without the user lifting a finger. However, where user intervention is required, the tool aims to make discovering security faults easy with the use of three outputs.

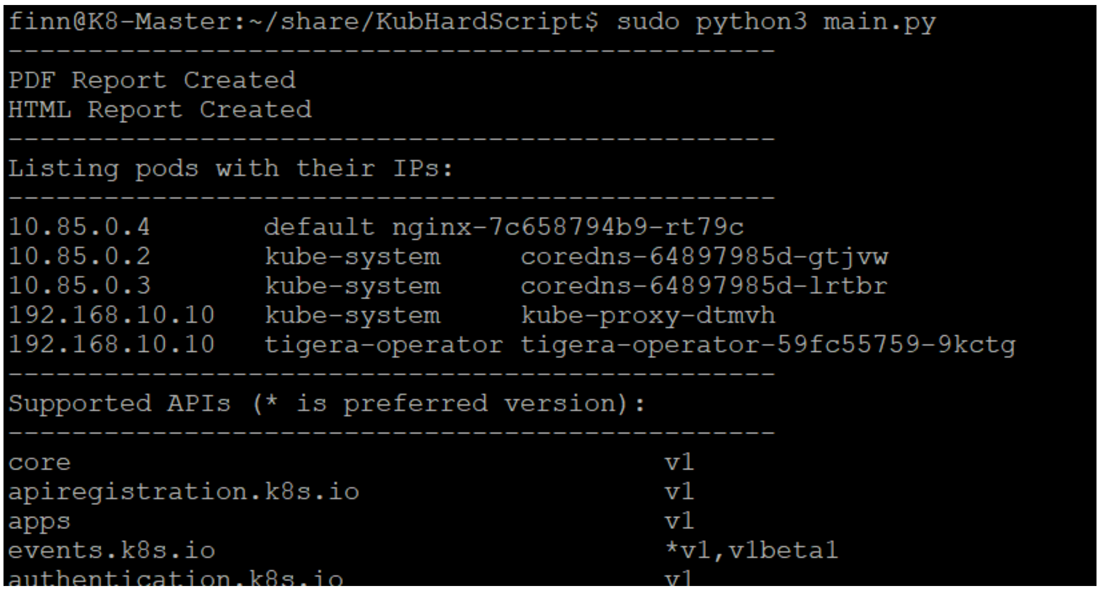
The opening checks performed by the tool relate to the permissions and file ownership of the Kubernetes configuration files. For example, the tool will inspect a YAML file confirming it has a greater permission level than 644. Next, the Kubernetes script utilises the checkov tool to perform static analysis on all the critical configuration files and saves this data. Finally, the script presents the information in three formats, including printing it to the CLI, a PDF report and an HTML report.

4.2.1 Checks

The Kubernetes hardening script performs over 15 permission checks on the most critical files relevant to Kubernetes environments. All of which are responsible for different Kubernetes resources. Furthermore, the script performs file ownership checks, ensuring they are set to root. The next set of checks managed by the script involves the utilisation of Chekov to inspect the Kubernetes configuration files. This static analysis utility has hundreds of inbuilt checks. The final scan made by the tool looks at the networking aspect of Kubernetes. A Nmap scan is run against the cluster IP address and the machine running the cluster.

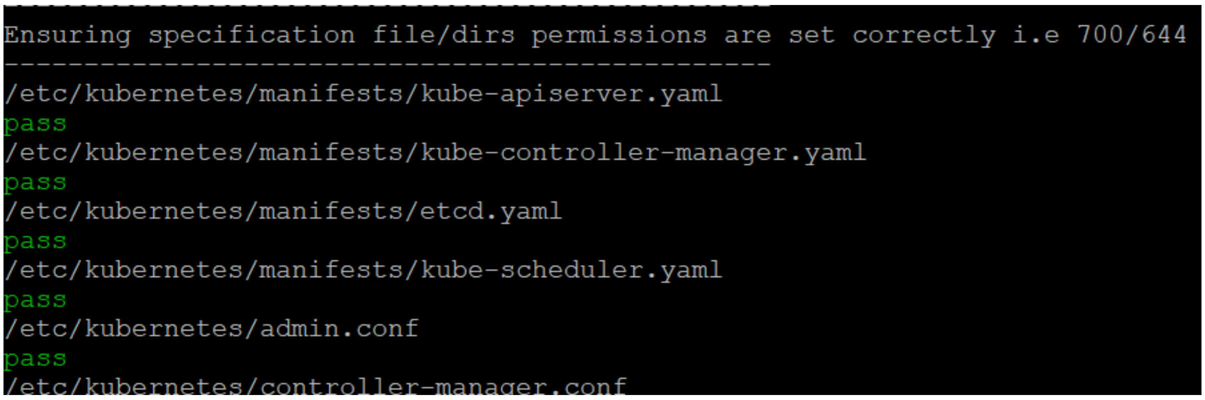
4.2.2 Output

The tool can output the results in three formats: the CLI, a PDF, and an HTML report. All of these have varying layouts and comprise different information. However, they ensure that the most critical aspects of the results are presented. When printed to the command line, several vital points unique to this format are displayed, including information such as the IP addresses of pods existing in the Kubernetes environment and the supported APIs. This can be seen below in **Figure 16** and **Figure 17.**



```
finn@K8-Master:~/share/KubHardScript$ sudo python3 main.py
-------------------------------------------------
PDF Report Created
HTML Report Created
-------------------------------------------------
Listing pods with their IPs:
-------------------------------------------------
10.85.0.4       default nginx-7c658794b9-rt79c
10.85.0.2       kube-system     coredns-64897985d-gtjvw
10.85.0.3       kube-system     coredns-64897985d-lrtbr
192.168.10.10   kube-system     kube-proxy-dtmvh
192.168.10.10   tigera-operator tigera-operator-59fc55759-9kctg
-------------------------------------------------
Supported APIs (* is preferred version):
-------------------------------------------------
core                                      v1
apiregistration.k8s.io                    v1
apps                                      v1
events.k8s.io                             *v1,v1beta1
authentication.k8s.io                     v1
```

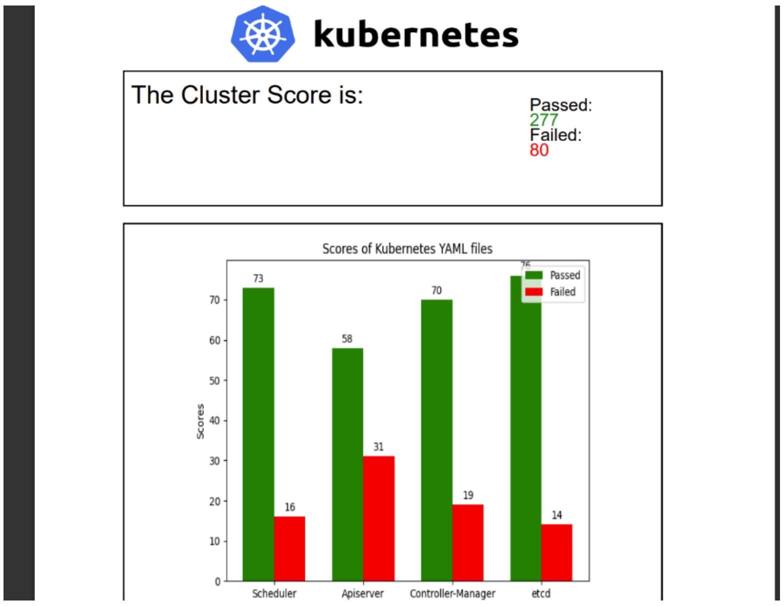
Figure 16 - Top Half of CLI Output



```
-------------------------------------------------
Ensuring specification file/dirs permissions are set correctly i.e 700/644
-------------------------------------------------
/etc/kubernetes/manifests/kube-apiserver.yaml
pass
/etc/kubernetes/manifests/kube-controller-manager.yaml
pass
/etc/kubernetes/manifests/etcd.yaml
pass
/etc/kubernetes/manifests/kube-scheduler.yaml
pass
/etc/kubernetes/admin.conf
pass
/etc/kubernetes/controller-manager.conf
```

Figure 17 - Sample of the Middle section of CLI Output

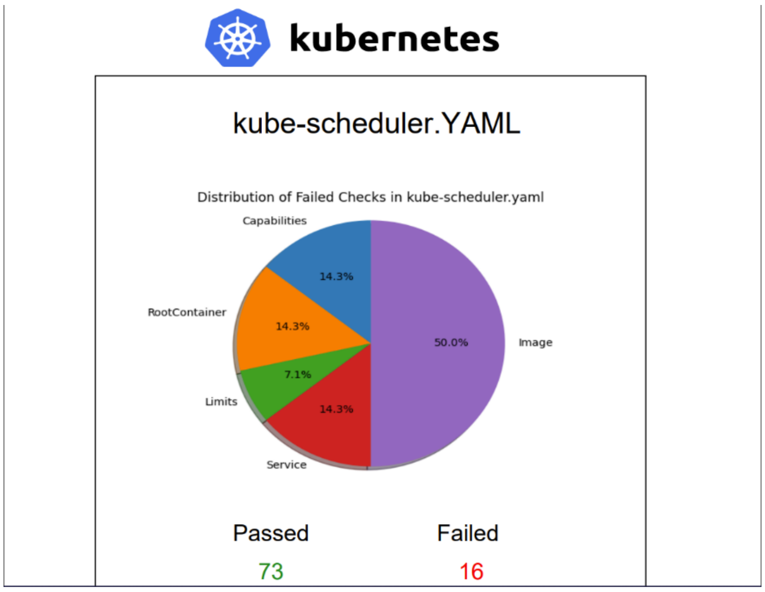
The PDF document presented below in **Figure 18** demonstrates the results from running the script on the test environment. Two hundred seventy-seven checks were performed at a passing grade, while 80 tests were failed. This is broken down even further in a bar chart comprising all the YAML files tested by the tool. Within the second page shown in **Figure 19**, users are given further information related to the class of failed checks in the form of a pie chart.


kubernetes

The Cluster Score is:

Passed: 277
Failed: 80

Figure 18 - PDF Document Summary Page


kubernetes

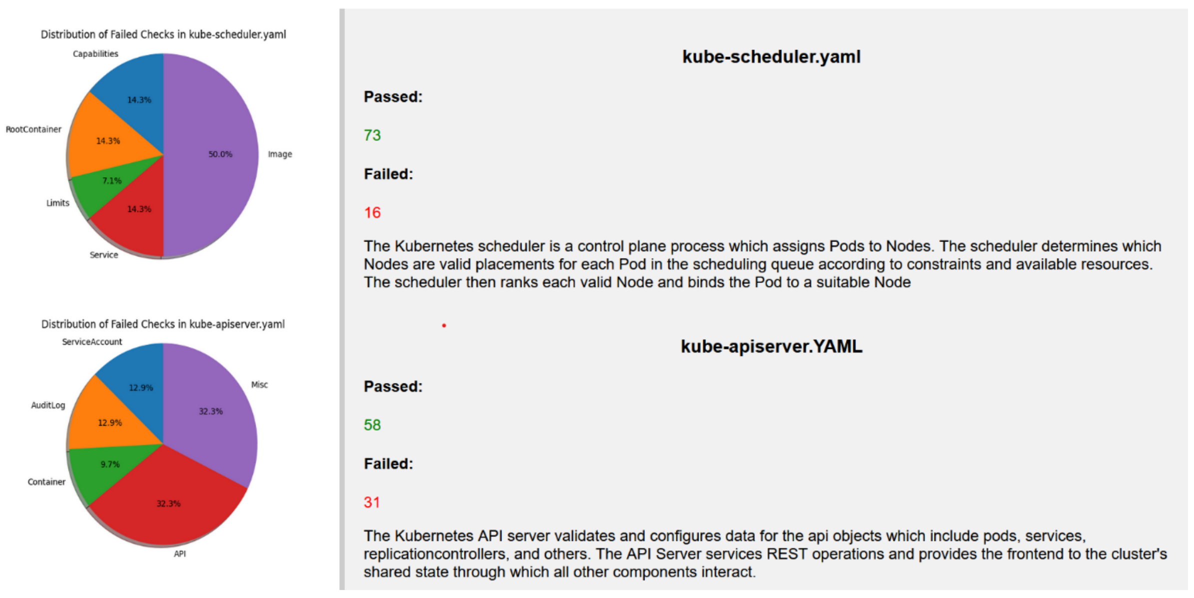
kube-scheduler.YAML

Passed 73

Failed 16

Figure 19 - PDF Document Kube Scheduler Page

The majority of the same information is included in the HTML report, as shown in **Figure 20**. However, it is structured slightly differently, opting for a design suited for an HTML document. In addition, this method of output includes a description of the Kubernetes services.



kube-scheduler.yaml

Passed:

73

Failed:

16

The Kubernetes scheduler is a control plane process which assigns Pods to Nodes. The scheduler determines which Nodes are valid placements for each Pod in the scheduling queue according to constraints and available resources. The scheduler then ranks each valid Node and binds the Pod to a suitable Node

kube-apiserver.YAML

Passed:

58

Failed:

31

The Kubernetes API server validates and configures data for the api objects which include pods, services, replicationcontrollers, and others. The API Server services REST operations and provides the frontend to the cluster's shared state through which all other components interact.

Figure 20 - Layout of the Main Section of the HTML Report

Chapter 5 Discussion

The following chapter will discuss and evaluate the creation of the Kubernetes hardening script. It will describe the thought process behind including different aspects and compare it to previous Kubernetes security applications. Furthermore, the effectiveness of the artefact and how it relates to the project's aim is considered, exploring where possible improvements could be made. Additionally, the chapter will explore the experiment with the static analysis tools, explaining what areas they could develop and the advantages of one. Again, this information will be related to the reviewed literature and the project's aims.

5.1 Static Analysis Tools

Each of the Kubernetes security tools tested had its positives and negatives. However, upon analysis of the results from the experiment described in 3.3.2 Kubernetes Security Tools it became apparent that checkov was the most suitable tool for the needs of the Kubernetes hardening script. This conclusion was brought about by many factors, including the familiarity with the language, having used python extensively before the project, and the range of checks it performed on the Kubernetes configuration files. Furthermore, the result metrics throughout testing were steady, and there exists a wealth of helpful documentation assisting users in their installation and utilisation of checkov. However, other effective static analysis tools were measured throughout the experiment, but they failed to meet the success criteria.

Only two were included in the static analysis tool experiment of the 4 Kubernetes security tools mentioned in the literature review. The others were deemed inadequate because they were fully-fledged security tools rather than basic static analysis applications.

Considering the experiment results, it is clear that the difference between the tools developed in the go programming language is negligible. Cases can be made for several tools to be known as the most efficient. For instance, Kube score has a particularly low memory usage. Moreover, these tools have low system time, CPU time and other low results. However, out of Kubsec and Kubescore, the software Kubescore edges it slightly due to Kubesecs comparatively large elapsed time metrics.

Checkov is the least efficient of the tools tested, with its relatively high results for all metrics apart from CPU usage. The tool's results remained steady throughout testing, but the first result was an outlier, as the data preceding this result had a massive decrease in all metrics. The difference between checkov and other tools can be attributed to the level of analysis and checks the tool provides. It covers a greater area, and the tool suffers from this taking longer than competitors. For example, checkov performs many checks related to authorisation, including but not limited to ensuring the authorisation-mode argument includes RBAC on all Kubernetes entities.

Table 3 Presents the results of investigating the static analysis tools and the checks they perform. It demonstrates that all tools provide some form of checks related to container security context. This means all tools inspect the config files searching for misconfigured privilege and access control settings for a Pod or Container. Security context settings include app armour and secomp.

Tool	*Auth Tests*	*Admission Controller Tests*	*Container Security Context*	*PSP's*	*NSP's*	*Ingress & Egress*	*Mutual TLS*	*Port Visibility*	*Policy Drift Detection*
Checkov	✓	✓	✓	x	x	x	x	x	x
Kubescore	x	x	✓	x	✓	x	x	x	x
Kubesec	x	x	✓	x	x	x	x	x	x
Kubelinter	x	x	✓	x	x	x	x	x	x
Kubeaudit	x	x	✓	x	x	x	x	x	x

Table 3 - Static Analysis Tools Check Investigation

While not only having the title of the most efficient tool, the Kubescore utility tests Kubernetes environments in an aspect that no other does, network policy checks. As presented below in **Figure 21** .Essentially Network policies are Kubernetes resources that control the traffic between pods and network endpoints, which means Kubescore checks if one exists and suggests creating one if the utility fails to find one. On the other hand, Checkov also provides a class of checks that other tools fail to include. This is not to mention the authorisation checks discussed previously in 2.3 Kubernetes Security Concerns and Issues but admission controller checks. Admission controllers are built-in security features that Kubernetes has to help govern and enforce how clusters are used. They function by intercepting requests sent to the API before the object is saved indefinitely but after the request has proper authorisation. The static analysis tool checkov provides substantially more security coverage when compared with its other competitors, as it includes authorisation, admission controller, and container context tests.

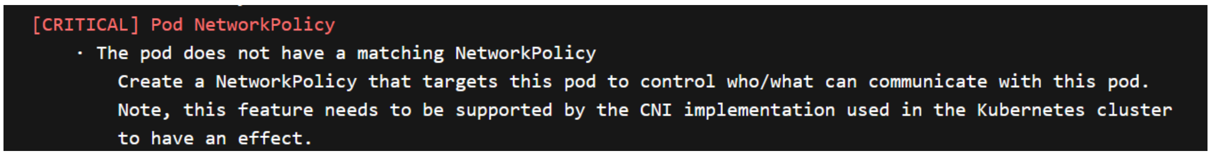

```
[CRITICAL] Pod NetworkPolicy
    · The pod does not have a matching NetworkPolicy
        Create a NetworkPolicy that targets this pod to control who/what can communicate with this pod.
        Note, this feature needs to be supported by the CNI implementation used in the Kubernetes cluster
        to have an effect.
```

Figure 21 - Kubescore Network Policy Check

The next aspect of the static analysis tools investigated related to the most effective method in presenting the results of a scan. Every tool tested throughout the experiment had a similar design and presentation as there is only so much a command-line interface can convey. However, utilities had their own set of quirks and points of interest. For instance, the Kubsec tool had the poorest output format, with its utilisation of a JSON type of layout printed through the CLI. Furthermore, the tool fails to include remediation suggestions, instead placing the onus on the user to search for solutions. Undeniably, the static analysis tool checkov is in contention for having the cleanest output while including essential attributes such as remediations and a link to further reading about the subject. The tool has some drawbacks in its extensive output that can clog up the CLI; however, this can be negated due to checkov offering a JSON format output.

As outlined in the aim and objectives, one of the primary purposes of this project was to conduct an experiment that measured the efficiency and resource usage of popular static analysis tools. The results from testing would assist in selecting an appropriate static analysis tool utilised in the Kubernetes hardening script. Selecting the appropriate tool for this task was influenced by several primary metrics including, range of security checks, output readability and contents, familiarisation with technologies and documentation availability. Overall, checkov is by far the best tool available to kubernetes users.

5.2 Kubernetes Hardening Script

Another primary objective of this project was the development of a Kubernetes hardening script. In the initial development stages of the script, it was concluded that python would be the most appropriate language to write the tool in. This decision has been proved correct due to the vast amounts of time saved, as new technologies or languages were avoided. In addition, the modules and external libraries utilised within the script saved time as well. If another programming language had been selected, the libraries and level of documentation would not be available when developing the tool.

The layout of the script with its unique file for individual components is, to a certain extent, the most effective method in structuring the Kubernetes script. It ensured the development was executed smoothly, as when programming the script, the process was iterative, allowing a feature to be added to the component file and tested. Furthermore, most Kubernetes best practice material was taken on a component to component basis.

The approach to developing the script leaves a lot to be desired. The tool does not use object-oriented program anywhere within its functionality. In future, when creating the script, how the code is structured and what programming methods are used will be given more thought. For instance, it might have been adequate to employ classes for different best practices within the Kubernetes resource files. A setting similar to ImagePullPolicyAlways would have a class and the checks would be located within it.

Best practices in developing program applications were followed in some cases. However, several lousy code practices are littered through the Kubernetes hardening script. One example of bad practice is the exclusion of properly commented code. If a developer were to attempt to improve the script or return to the code after a long absence, it would be hard to understand the code initially.

The tool developed in this project builds on the current progress made by other Kubernetes security tools. For example, it incorporates ideas from Kubesec, such as a section that gives the user information about their Kubernetes environment. Like the proof of concept script described in the literature review that demonstrated what an integrated testing suite would look like. The Kubernetes security tool developed in this project performs a Nmap scan on relevant IP addresses.

5.2.1 Checks

Permission checks on Kubernetes resource files ensuring they meet a minimum level of permission is a feature that no other of the tested tools incorporates. Including the POC script described in the literature review and the various other tools mentioned. Furthermore, file ownership is another set of checks that, from research, no other tool apart from this one has the ability to perform. Both sets of checks are crucial in ensuring best practices are followed. In the case of maintaining appropriate permission levels, it ensures the integrity of the file. In addition, the file should be writable by only the administrators on the system, which also applies to the file ownership checks.

As described in the Results section, with the utilisation of Checkov, the tool provides three classes of checks. Authorisation, admission controller and container security context are Kubernetes aspects that are checked by the developed tool, which is a decent amount of coverage compared with the previous tools. However, a large area of a Kubernetes environment fails to be covered. For instance, A Pod Security Policy is a cluster-level resource that controls security sensitive aspects of the pod specification. They are important in maintaining a secure Kubernetes environment; therefore, tests on these resources are beneficial. However, the tool does provide admission controller checks which are the replacement for pod security policies. Furthermore, it is good practice to include checks in a security tool that looks at previous technologies or deprecated functionality as it is highly likely they will still be employed in real-world environments. Therefore, apart from Kubesec providing checks for network security policies, the developed tool covers more security aspects than all the previous tools described in the literature review and used in the tools experiment.

Finally, there are serval pieces of code that are messy and nowhere near as efficient as they could be. This is evident in **Figure 8**, with the code responsible for checking for the appropriate level of permissions. If permissions are not at the level required, the program will set every file to the correct permissions rather than that select file. Again, this would impact the efficiency of the script, even if it was minimal.

5.2.2 Output

Creating a tool with a readable and clean output was one of the goals of this project. Three methods are available to users and offer varying levels of information that can be used to assist in the security of their Kubernetes environment. The CLI followed a similar design to its competitors, with the utilisation of colour for results helping improve the readability. Furthermore, the output layout through the command line is done sequentially and uses lines to break results into sections. Finally, the tool includes an information section related to the Kubernetes environment with data such as the latest releases of Kubernetes software and the IP address of available pods.

The pdf document is another method of output available to users and effectively conveys the essential information of the scan. However, the design is lacking, and with more time, it would be beneficial to implement a remediation aspect to the file, assisting users in fixing the discovered faults. Notably, the utilisation of frames to break down the different aspects of the scan is effective, but this would be improved on and implemented across all pages with more time. Furthermore, no other tool tested provided a document of results similar to the one implemented in this tool. Finally, similar to the HTML report, a description of the resource file would improve users' knowledge and help them understand the role the file plays.

The best method of output created by the tool is the HTML report due to reasons including the description of Kubernetes resources as previously mentioned. Furthermore, the structure and style of the HTML file are clean, with the utilisation of section bars for each file and the placement of the pie charts. In its current state, the design is not awful. However, specific improvements could be made through colour coordination and the employment of tables, including a page that outlines the necessary remediation needed.

Chapter 6 Conclusion and Future Work

6.1 Future Work

Due to the continual improvements and new releases of Kubernetes security tools, tools could be developed that incorporate even more than three classes of checks. As a result, a Kubernetes tool has the potential to become a fully-fledged testing framework akin to a product such as Metasploit. Should this occur, it would be beneficial to conduct testing on this application. Furthermore, it would be interesting to test Kubernetes static analysis tools and all tools related to securing Kubernetes environments. Implementing this in the Kubernetes hardening script provides further benefits that a static analysis tool cannot achieve. Improvements could also be made in testing methods; for instance, several insecure clusters could be configured with varying security faults injected. The tools would scan these environments, and the effectiveness would be calculated by looking at the total and range of faults discovered.

If the constraint of time is not so severe, the scope of the Kubernetes hardening script could be expanded in many ways. For instance, while checkov is a competent statics analysis software, it leaves a lot to be desired with the exclusion of PSP and NSP checks. Therefore, a new static analysis tool or the functionality of one could be developed that covers the aspects that checkov and all others fail to include. Furthermore, the script has the potential to harden not only Kubernetes but other software related to containerisation. Finally, the tool could be configured to operate as a proper security tool with the addition of switches, flags and appropriate resource bundling.

6.2 Conclusion

In conclusion, this paper aimed to complete an examination of Kubernetes security by developing a hardening script and conducting an experiment on Kubernetes static analysis tools. The project determined that on exploring current material, tools, and software security, even with the vast progress made, Kubernetes security is exceptionally fragile in some aspects.

This project's development of a Kubernetes hardening script has shown successfully that existing Kubernetes security tools can be improved on in many ways, including the type of checks they perform and the user experience. The final implementation of the script provided a tool capable of executing serval checks on Kubernetes environments, ensuring they adhere to best practices set by security professionals. Checks included in the tool range from permission related to the contents of Kubernetes resource files. Apart from a tool that competently issues checks and provides remediation advice, the tool met the objective of utilising a clean method of output, where the tool excelled in. Although several limitations exist in the script related to the lack of coverage in some Kubernetes components, such as the control plane, the incorporation of checkov and other checking methods ensures the tool remains highly effective. As a result, this tool could be used by, with more development, organisations previously discussed in the introduction of the paper to ensure their Kubernetes environment is as secure as possible.

The features required to make a tool effective were identified through the experimentation involving the selected static analysis tools. Measuring the resource usage and efficiency of the tools helped determine what utility was most potent in certain areas, ultimately assisting in choosing the appropriate tool utilised in the hardening script. The results demonstrated that checkov was by some distance the most effective tool but not the most efficient. That title belonged to Kubescore with its low metrics across the board and particularly in the amount of memory it utilises. Considering the readability of a tool's output was subjective, there was little between the tested tools in the form of an outlying tool with a standout output. However, specific tools including Kubesec and Kubescore had a few defining features that set it apart. The testing and research undertaken in this report have highlighted that there are no perfect Kubernetes security tools, and there is room for improvement. Furthermore, while static analysis tools are suitable for inspecting Kubernetes resource files, they fail to attempt to exploit the discovered faults, limiting the amount of remediation information they can provide to users.

Regarding the research question outlined in the introductory section of the paper, the paper has furtherly explored Kubernetes security by completing the objectives. An Examination of the most significant issues pertaining to Kubernetes-based environmental security was carried out initially through the research phase. After this, the tools were tested, where the security faults and the exploits caused by these issues were learned thoroughly, including the internal settings of Kubernetes. Furthermore, experimentation of the tools demonstrated that the current tools leave a lot to be desired, and this will pose a problem when improving security in a Kubernetes cluster. Finally, the script's development helped conclude that there is a vast attack surface available to malicious users looking to gain unauthorised access. When searching for material in developing the script, there was not a substantial amount of security-based material that could assist security practitioners.

List of References

Artem, L., Tetiana, B., Larysa, M. and Vira, V., 2020. Eliminating privilage escalation to root in containers running on Kubernetes. *Scientific and practical cyber security journal*.

Budigiri, G., Baumann, C., Mühlberg, J.T., Truyen, E. and Joosen, W., 2021, June. Network policies in kubernetes: Performance evaluation and security analysis. In *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)* (pp. 407-412). IEEE.

Chen, J., Feng, Z., Wen, J.Y., Liu, B. and Sha, L., 2019, March. A container-based DoS attack-resilient control framework for real-time UAV systems. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 1222-1227). IEEE.

Chen, J., Sasson, A. and Zelivansky, A., 2022. Unsecured Kubernetes Instances Could Be Vulnerable to Exploitation. [online] Unit42. Available at: <https://unit42.paloaltonetworks.com/unsecured-kubernetes-instances/> [Accessed 13 April 2022].

Docs.python.org. n.d. *subprocess — Subprocess management — Python 3.10.4 documentation*. [online] Available at: <https://docs.python.org/3/library/subprocess.html> [Accessed 17 May 2022].

E Hecht, L., 2018. What the Data Says about Kubernetes Deployment Patterns. [online] Thenewstack.io. Available at: <https://thenewstack.io/data-says-kubernetes-deployment-patterns/> [Accessed 17 May 2022].

Fowley, T., 2021. Security of Virtual Infrastructures: Assessing Kubernetes Attack Automation.

Gummaraju, J., Desikan, T. and Turner, Y., 2015. Over 30% of official images in docker hub contain high priority security vulnerabilities. Technical Report.

Henriksson, O. and Falk, M., 2017. Static vulnerability analysis of docker images.

JetBrains. n.d. PyCharm: the Python IDE for Professional Developers by JetBrains. [online] Available at: <https://www.jetbrains.com/pycharm/> [Accessed 17 May 2022].

Jian, Z. and Chen, L., 2017, March. A defense method against docker escape attack. In Proceedings of the 2017 International Conference on Cryptography, Security and Privacy (pp. 142-146)

Kubernetes. n.d. adidas Case Study. [online] Available at: <https://kubernetes.io/case-studies/adidas/> [Accessed 17 May 2022].
Kubernetes. n.d. Case Studies. [online] Available at: <https://kubernetes.io/case-studies/> [Accessed 17 May 2022].
Kubernetes. 2021. Client Libraries. [online] Available at: <https://kubernetes.io/docs/reference/using-api/client-libraries/> [Accessed 17 May 2022].
Kubernetes Security Practices. In 2020 IEEE Secure Development (SecDev) (pp. 58-64). IEEE.
Matplotlib.org. n.d. *Matplotlib — Visualization with Python*. [online] Available at: <https://matplotlib.org/> [Accessed 17 May 2022].
Minna, F., Blaise, A., Rebecchi, F., Chandrasekaran, B. and Massacci, F., 2021. Understanding the Security Implications of Kubernetes Networking. IEEE Security & Privacy, 19(5), pp.46-56.
Mytilinakis, P., 2020. *Attack methods and defenses on Kubernetes* (Master's thesis, Πανεπιστήμιο Πειραιώς).
Nam, J., Lee, S., Seo, H., Porras, P., Yegneswaran, V. and Shin, S., 2020. {BASTION}: A security enforcement network stack for container networks. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)* (pp. 81-95).
NIST. 2022. NIST. [online] Available at: <https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-190.pdf> [Accessed 25 March 2022].
Prizmant, D., 2022. *Siloscape: First Known Malware Targeting Windows Containers to Compromise Cloud Environments*. [online] Unit42. Available at: <https://unit42.paloaltonetworks.com/siloscape/> [Accessed 18 March 2022].
PyPI. n.d. termcolor. [online] Available at: <https://pypi.org/project/termcolor/> [Accessed 17 May 2022].
Rangta, M., 2022. Tools for Security Auditing and Hardening in Microservices Architecture
Red Hat blog. 2021. The State of Kubernetes Security. [online] Available at: <https://www.redhat.com/en/blog/state-kubernetes-security> [Accessed 17 May 2022].

Reeves, M., Tian, D.J., Bianchi, A. and Celik, Z.B., 2021, October. Towards Improving Container Security by Preventing Runtime Escapes. In *2021 IEEE Secure Development Conference (SecDev)* (pp. 38-46). IEEE.

Reportlab.com. 2020. *Report Lab User Guide*. [online] Available at: <https://www.reportlab.com/docs/reportlab-userguide.pdf> [Accessed 17 May 2022].

SentinelOne. 2022. *DarkRadiation* | *Abusing Bash For Linux and Docker Container Ransomware*. [online] Available at: <https://www.sentinelone.com/blog/darkradiation-abusing-bash-for-linux-and-docker-container-ransomware/> [Accessed 18 March 2022].

Shamim, M.S.I., Bhuiyan, F.A. and Rahman, A., 2020, September. XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to

Sultan, S., Ahmad, I. and Dimitriou, T., 2019. Container security: Issues, challenges, and the road ahead. *IEEE Access*, *7*, pp.52976-52996.

Team, R., Higashi, M., Kumar, G., Team, R. and Chiodi, M., 2018. *Lessons from the Cryptojacking Attack at Tesla*. [online] RedLock. Available at: <https://redlock.io/blog/cryptojacking-tesla> [Accessed 17 May 2022].

Tigera.io. 2022. *Project Calico*. [online] Available at: <https://www.tigera.io/project-calico/> [Accessed 22 February 2022].

Towett, E., 2022. *How to set up Kubernetes Cluster on Ubuntu 20.04 with kubeadm and CRI-O*. [online] Citizix – Devops, Cloud, Infrastructure and Tech Content. Available at: <https://citizix.com/how-to-set-up-kubernetes-cluster-on-ubuntu-20-04-with-kubeadm-and-cri-o/> [Accessed 17 May 2022].

Vailshery, L., 2022. *Kubernetes use challenges worldwide 2021* | *Statista*. [online] Statista. Available at: <https://www.statista.com/statistics/1234145/kubernetes-usage-challenges-organizations/> [Accessed 17 May 2022].

Virtualbox.org. n.d. *Chapter�4.�Guest Additions*. [online] Available at: <https://www.virtualbox.org/manual/ch04.html> [Accessed 17 May 2022].

Bibliography

Akula, M., Li, V. and Cotton, T., 2021. Practical Kubernetes Security Learning using Kubernetes Goat.

Armstrong, J., 2022. Calico in 2020: The World’s Most Popular Kubernetes Networking and Security solution. [online] Tigera.io. Available at: <https://www.tigera.io/blog/calico-in-2020-the-worlds-most-popular-kubernetes-cni/> [Accessed 22 February 2022].

Beale, J., 2022. *Attacking and Defending Kubernetes: Bust-A-Kube – Episode 1 | InGuardians*. [online] Inguardians.com. Available at: <https://www.inguardians.com/attacking-and-defending-kubernetes-bust-a-kube-episode-1/> [Accessed 25 March 2022].

Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J., 2016. Borg, omega, and kubernetes. Communications of the ACM, 59(5), pp.50-57

Checkov.io. n.d. *checkov*. [online] Available at: <https://www.checkov.io/> [Accessed 17 May 2022].

GitHub. n.d. *GitHub - stackrox/kube-linter: KubeLinter is a static analysis tool that checks Kubernetes YAML files and Helm charts to ensure the applications represented in them adhere to best practices.*. [online] Available at: <https://github.com/stackrox/kube-linter> [Accessed 17 May 2022].

GitHub. n.d. *GitHub - zegl/kube-score: Kubernetes object analysis with recommendations for improved reliability and security*. [online] Available at: <https://github.com/zegl/kube-score> [Accessed 17 May 2022].

GitHub. n.d. *GitHub - Shopify/kubeaudit: kubeaudit helps you audit your Kubernetes clusters against common security controls*. [online] Available at: <https://github.com/Shopify/kubeaudit> [Accessed 17 May 2022].

Kubesec.io. 2020. *kubesec.io :: kubesec.io*. [online] Available at: <https://kubesec.io/> [Accessed 17 May 2022].

Lin, X., Lei, L., Wang, Y., Jing, J., Sun, K. and Zhou, Q., 2018, December. A measurement study on linux container security: Attacks and countermeasures. In *Proceedings of the 34th Annual Computer Security Applications Conference* (pp. 418-429).

Reshetova, E., Karhunen, J., Nyman, T. and Asokan, N., 2014, October. Security of OS-level virtualization technologies. In Nordic Conference on Secure IT Systems (pp. 77-93). Springer, Cham.

Appendices

Main.py

```
from kubernetes import client, config
from termcolor import colored
import subprocess
import MasterNode
import ReportHTML
import Report
import WorkerNodes

import etcd
from NMAP import Nmap
from etcd import etcd
import sys
import subprocess
import pkg_resources

def info():
    config.load_kube_config()
    v1 = client.CoreV1Api()
    print("Listing pods with their IPs:")
    print("--------------------------------------------------")
    ret = v1.list_pod_for_all_namespaces(watch=False)
    for i in ret.items:
        print("%s\t%s\t%s" % (i.status.pod_ip, i.metadata.namespace,
i.metadata.name))

    print("--------------------------------------------------")
    print("Supported APIs (* is preferred version):")
    print("--------------------------------------------------")
    print("%-40s %s" %
          ("core", ",".join(client.CoreApi().get_api_versions().versions)))
    for api in client.ApisApi().get_api_versions().groups:
        versions = []
        for v in api.versions:
            name = ""
            if v.version == api.preferred_version.version and len(
                    api.versions) > 1:
                name += "*"
            name += v.version
            versions.append(name)
        print("%-40s %s" % (api.name, ",".join(versions)))

def hardening():
    print("test")




def requirments():
    required = {'python-nmap', 'kubernetes', 'PyYAML', 'termcolor'}
    installed = {pkg.key for pkg in pkg_resources.working_set}
    missing = required - installed

    if missing:
        python = sys.executable
        subprocess.check_call([python, '-m', 'pip', 'install', *missing],
stdout=subprocess.PIPE)
```

```
def main():
    requirments()
    info()
    MasterNode.main()
    etcd()
    WorkerNodes.main()
    Nmap()
    ReportHTML
    Report



main();
```

MasterNode.py

```
from kubernetes import client, config
from termcolor import colored
import array
import subprocess
import json
import yaml
import os


cmds = "/etc/kubernetes/manifests/kube-apiserver.yaml", \
       "/etc/kubernetes/manifests/kube-controller-manager.yaml", \
       "/etc/kubernetes/manifests/etcd.yaml",
"/etc/kubernetes/manifests/kube-scheduler.yaml", \
       "/etc/kubernetes/admin.conf","/etc/kubernetes/controller-
manager.conf", \
       "/etc/kubernetes/pki/apiserver.crt", "/etc/kubernetes/pki/apiserver-
etcd-client.crt", "/etc/kubernetes/pki/apiserver-kubelet-client.crt",
"/etc/kubernetes/pki/ca.crt", "/etc/kubernetes/pki/front-proxy-ca.crt",
"/etc/kubernetes/pki/front-proxy-client.crt"
cmdsEtcd = "/etc/kubernetes/pki/etcd"
array.test = []
pas = ""

def permissions():
    # array.stdout = []
    print("----------------------------------------------------")
    print("Ensuring specification file/dirs permissions are set correctly
i.e 700/644")
    print("----------------------------------------------------")
    for cmd in cmds:
        PodPermissions = subprocess.run(["stat", "-c", "%a", cmd],
stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                                        text=True)
        output = (int(PodPermissions.stdout))
        test = output
        print(cmd)

        if test >= 644:
            pas = "pass"
            print(colored(pas, 'green'))

        elif test <= 600:
            pas = "fail"
            print(colored(pas, 'red'))
            for cmd in cmds:
                PodPermissions = subprocess.run(["chmod", "644", cmd],
stdout=subprocess.PIPE,
                                                stderr=subprocess.PIPE,
text=True)
                # PodPermissions = (int(PodPermissions.stdout))
                # print(PodPermissions)

    etcd = subprocess.run(["stat", "-c", "%a", cmdsEtcd],
stdout=subprocess.PIPE,
                          stderr=subprocess.PIPE, text=True)
    output = int(etcd.stdout)
    print(cmdsEtcd)
    if output >= 700:
        pas = "pass"
        print(colored(pas, 'green'))

    elif output < 700:
        pas = "fail"
        print(colored(pas, 'red'))
        subprocess.run(["chmod", "700", cmdsEtcd], stdout=subprocess.PIPE,
                       stderr=subprocess.PIPE, text=True)



def fileOwnership():
    print("-------------------------------------------------")
    print("Ensuring that the specification file ownership is set to
root:root.")
    print("-------------------------------------------------")


    for cmd in cmds:
        PodfileOwner = subprocess.run(["stat", "-c", "%U:%G", cmd],
stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                                        text=True)
        output = (str(PodfileOwner.stdout))
        output = output.strip()
        print(cmd)

        if output == "root:root":
            pas = "pass"
            print(colored(pas, 'green'))

        else:
            pas = "fail"
            print(colored(pas, 'red'))
            for cmd in cmds:
                PodPermissions = subprocess.run(["chown", "root:root",
cmd], stdout=subprocess.PIPE,
                                                stderr=subprocess.PIPE)
                # PodPermissions = (int(PodPermissions.stdout))
                # print(PodPermissions)
    etcd = subprocess.run(["stat", "-c", "%U:%G", cmdsEtcd],
stdout=subprocess.PIPE,
                          stderr=subprocess.PIPE, text=True)
    output = etcd.stdout
    output = output.strip()
    print(cmdsEtcd)
    if output == "root:root":
        pas = "pass"
        print(colored(pas, 'green'))

    else:
        pas = "fail"
        print(colored(pas, 'red'))
        subprocess.run(["chown", "root:root", cmdsEtcd],
stdout=subprocess.PIPE,
                       stderr=subprocess.PIPE, text=True)




def ApiYaml():
    with open('kube-apiserver.json', "w") as outfile:
        test = subprocess.run(["checkov", "--compact", "--quiet", "-o",
"json", "-f", "/etc/kubernetes/manifests/kube-apiserver.yaml"],
stdout=outfile, stderr=subprocess.PIPE,
                                    text=True)

    with open('kube-apiserver.json', 'r') as json_file:
        json_load = json.load(json_file)

    print("-------------------------------------------------")
    print("Running checks on the kube-apiserver.yaml file")
    print("-------------------------------------------------")
    passed = (json_load['summary']['passed'])
    failed = (json_load['summary']['failed'])

    print("Passed:")
    print(colored(passed, 'green'))
    print("Failed:")
    print(colored(failed, 'red'))
    print(colored("Full details and remadations can be found in kube-
apiserver.json", 'yellow'))

def KCMYaml():
    with open('kube-controller-manager.json', "w") as outfile:
        test = subprocess.run(["checkov", "--compact", "--quiet", "-o",
"json", "-f", "/etc/kubernetes/manifests/kube-controller-manager.yaml"],
stdout=outfile, stderr=subprocess.PIPE,
                                    text=True)

    with open('kube-controller-manager.json', 'r') as json_file:
        json_load = json.load(json_file)

    print("-------------------------------------------------")
    print("Running checks on the kube-controller-manager.yaml file")
    print("-------------------------------------------------")
    passed = (json_load['summary']['passed'])
    failed = (json_load['summary']['failed'])

    print("Passed:")
    print(colored(passed, 'green'))
    print("Failed:")
    print(colored(failed, 'red'))
    print(colored("Full details and remadations can be found in kube-
controller-manager.json", 'yellow'))

def SchedYaml():

    with open('kube-scheduler.json', "w") as outfile:
        test = subprocess.run(["checkov", "--compact", "--quiet", "-o",
"json", "-f", "/etc/kubernetes/manifests/kube-scheduler.yaml"],
stdout=outfile, stderr=subprocess.PIPE,
                                    text=True)

    with open('kube-scheduler.json', 'r') as json_file:
        json_load = json.load(json_file)

    print("------------------------------------------------")
    print("Running checks on the kube-scheduler.yaml file")
    print("------------------------------------------------")
    passed = (json_load['summary']['passed'])
    failed = (json_load['summary']['failed'])

    print("Passed:")
    print(colored(passed, 'green'))
    print("Failed:")
    print(colored(failed, 'red'))
    print(colored("Full details and remadations can be found in kube-
scheduler.json", 'yellow'))

def main():
    permissions()
    fileOwnership()
    ApiYaml()
    KCMYaml()
    SchedYaml()
if __name__ == '__main__':
    main()
```

Nmap.py

```
from kubernetes import client, config
import nmap
import socket


def Nmap():
    config.load_kube_config()
    api = client.CoreV1Api()
    service = api.read_namespaced_service(name="kubernetes",
namespace="default")
    print("------------------------------------------------")
    print("Running nmap scan on cluster machine")
    print("------------------------------------------------")
    print("cluster ip:" + service.spec.cluster_ip)
    print("------------------------------------------------")
    fin = service.spec.cluster_ip

    hostname = socket.gethostname()
    IPADDR = socket.gethostbyname(hostname)

    nm = nmap.PortScanner()

    scan_range = nm.scan(hosts=IPADDR)

    nm.all_hosts()

    for host in nm.all_hosts():
        print("Host: %s(%s)" % (host, nm[host].hostname()))

        print("Open TCP Ports: ")

        print("%s" % (nm[host].all_tcp()))

        print("Open UDP Ports: ")

        print("%s" % (nm[host].all_udp()))

        print("state: ")

        print("%s" % (nm[host].state()))

        print("%s" % (nm[host].all_protocols()))

    nm2 = nmap.PortScanner()

    scan_range = nm2.scan(hosts=fin)


    nm2.all_hosts()

    for host in nm2.all_hosts():
        print("Host: %s(%s)" % (host, nm2[host].hostname()))

        print("Open TCP Ports: ")

        print("%s" % (nm2[host].all_tcp()))

        print("Open UDP Ports: ")

        print("%s" % (nm2[host].all_udp()))

        print("state: ")

        print("%s" % (nm2[host].state()))

        print("%s" % (nm2[host].all_protocols()))


print("--------------------------------------------------")
print("PDF Report Created")
print("HTML Report Created")
print("--------------------------------------------------")
def main():
    Nmap()
```

```
if __name__ == '__main__':
    main()
```

Report.py

```
import json
from reportlab.pdfgen.canvas import Canvas
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib.units import inch
from reportlab.platypus import Paragraph, Frame
from reportlab.platypus import Image
from reportlab.lib.styles import ParagraphStyle
from reportlab.pdfbase import pdfmetrics
from reportlab.lib.colors import HexColor
from time import time, ctime


json1 = "kube-scheduler.json", "kube-apiserver.json", "kube-controller-
manager.json", "etcd.json"
t = time()
timeM = ctime(t)

style = ParagraphStyle(
    name='Normal',
    fontSize=20,
)
stylePassed = ParagraphStyle(
    name='Normal',
    fontSize=14,
    textColor=(HexColor('#038a27')),
    # leading=15,
    leftIndent=320,
)
styleFailed = ParagraphStyle(
    name='Normal',
    fontSize=14,
    textColor=(HexColor('#FF0000')),
    leftIndent=320,
)
stylePassedH = ParagraphStyle(
    name='Normal',
    fontSize=14,
    leftIndent=320,
)
styleFailedH = ParagraphStyle(
    name='Normal',
    fontSize=14,
    leftIndent=320,
)




style1 = ParagraphStyle(
    name='Normal',
    fontSize=40,
    leftIndent=25,
)
```

```
style2 = ParagraphStyle(
    name='Normal',
    fontSize=12,
    leftIndent=210,
)

style3 = ParagraphStyle(
    name='Normal',
    fontSize=17,
)

style4 = ParagraphStyle(
    name='Normal',
    fontSize=12,
    leading=15,
    spaceBefore=17,
    textColor=(HexColor('#038a27')),
)
style5 = ParagraphStyle(
    name='Normal',
    fontSize=11,
    leading=18,
    spaceBefore=17,
)

style6 = ParagraphStyle(
    name='Normal',
    fontSize=12,
    leading=15,
    spaceBefore=40,
    textColor=(HexColor('#e62727')),
)

styleSchedH = ParagraphStyle(
    name='Normal',
    fontSize=15,
    leftIndent=20,


)
styleSchedP = ParagraphStyle(
    name='Normal',
    fontSize=13,
    leftIndent=50,
    textColor=(HexColor('#038a27')),



)
styleSchedF = ParagraphStyle(
    name='Normal',
    fontSize=13,
    leftIndent=50,
    textColor=(HexColor('#FF0000')),



)
```

```
styles = getSampleStyleSheet()
story = []
story1 = []
story2 = []
story3 = []
story4 = []
story5 = []
story6 = []
story7 = []
story10 = []
story11 = []
story12 = []
story13 = []
story14 = []

passed = []
failed = []
sum2 = []
image = "kubernetes-horizontal-color.png"
image2 = "kube-apiserver.png"
image3 = "kube-scheduler.png"
image4 = "kube-controller-manager.png"
image5 = "etcd.png"

for j in json1:
    with open(j, 'r') as json_file:
        json_load = json.load(json_file)

    passedJ = (json_load['summary']['passed'])
    passed.append(passedJ)
for j in json1:
    with open(j, 'r') as json_file:
        json_load = json.load(json_file)

    failedJ = (json_load['summary']['failed'])
    failed.append(failedJ)

for j in json1:
    with open(json1[0], 'r') as json_file:
        json_load = json.load(json_file)

    sum1 = (json_load['summary'])

    sum2.append(sum1)





passedT = sum(passed)
failedT = sum(failed)

story.append(Paragraph("The Cluster Score is:", style))
story.append(Paragraph("Passed:", stylePassedH))
story.append(Paragraph(str(passedT), stylePassed))
story.append(Paragraph("Failed:", styleFailedH))
story.append(Paragraph(str(failedT), styleFailed))
story11.append(Image('Summary.png',  width=330, height=305))
# story10.append(Paragraph("kube-scheduler.YAML", styleSchedH))
# story10.append(Paragraph(str(passed[0]), styleSchedP))
# story10.append(Paragraph(str(failed[0]), styleSchedF))
# story10.append(Paragraph("kube-scheduler.YAML", styleSchedH))
# story10.append(Paragraph(str(passed[1]), styleSchedP))
# story10.append(Paragraph(str(failed[1]), styleSchedF))

t = time()
t2 = str(t)

c = Canvas('PDF_Report' + t2 + '.pdf')
c.setFont("Helvetica", 16)
c.drawString(100, 245, "kube-scheduler.YAML")
c.setFillColor(HexColor('#038a27'))
c.drawString(120, 227, str(passed[0]))
c.setFillColor(HexColor('#FF0000'))
c.drawString(200, 227, str(failed[0]))

c.setFillColor(HexColor('#000000'))

c.drawString(100, 200, "kube-apiserver.YAML")
c.setFillColor(HexColor('#038a27'))
c.drawString(120, 182, str(passed[1]))
c.setFillColor(HexColor('#FF0000'))
c.drawString(200, 182, str(failed[1]))

c.setFillColor(HexColor('#000000'))

c.drawString(100, 155, "kube-controller-manager.YAML")
c.setFillColor(HexColor('#038a27'))
c.drawString(120, 137, str(passed[2]))
c.setFillColor(HexColor('#FF0000'))
c.drawString(200, 137, str(failed[2]))

c.setFillColor(HexColor('#000000'))

c.drawString(100, 110, "etcd.YAML")
c.setFillColor(HexColor('#038a27'))
c.drawString(120, 92, str(passed[3]))
c.setFillColor(HexColor('#FF0000'))
c.drawString(200, 92, str(failed[3]))

c.setFillColor(HexColor('#038a27'))
c.drawString(370, 200, "Passed")
c.setFillColor(HexColor('#FF0000'))
c.drawString(370, 150, "Failed")
c.setFont("Courier", 10)
c.setFillColor(HexColor('#000000'))
c.drawCentredString(4 * inch, 0.2 * inch, timeM)

f = Frame(1 * inch, 9.4 * inch, 6 * inch, 1.5 * inch, showBoundary=1)
f1 = Frame(1 * inch, 4.2 * inch, 6 * inch, 5 * inch, showBoundary=1)
f3 = Frame(1 * inch, 0.5 * inch, 6 * inch, 3.5 * inch, showBoundary=1)


f.addFromList(story, c)
f.addFromList(story3, c)
f.addFromList(story4, c)
f1.addFromList(story1, c)
f3.addFromList(story2, c)
f1.addFromList(story6, c)
f3.addFromList(story5, c)
f1.addFromList(story7, c)
```

```
f1.addFromList(story11, c)
f3.addFromList(story10, c)
f3.addFromList(story12, c)



c.drawImage(image, 2.2 * inch, 11 * inch, width=230, height=45)

c.showPage()




f4 = Frame(1 * inch, 0.9 * inch, 6 * inch, 10 * inch, showBoundary=1)
f4.addFromList(story, c)
c.setFont("Helvetica", 23)
c.drawString(180, 740, "kube-scheduler.YAML")
c.drawImage(image, 2.2 * inch, 11 * inch, width=230, height=45)
c.drawImage(image3, 1.15 * inch, 6.1 * inch, width=400, height=275)


# c.setFillColor(HexColor('#000000'))



c.setFont("Helvetica", 18)
c.drawString(180, 420, "Passed")
c.setFillColor(HexColor('#038a27'))
c.drawString(200, 390, str(passed[0]))

c.setFillColor(HexColor('#000000'))
c.drawString(340, 420, "Failed")
c.setFillColor(HexColor('#000000'))
c.setFillColor(HexColor('#FF0000'))
c.drawString(357, 390, str(failed[0]))
c.setFillColor(HexColor('#000000'))
c.setFont("Helvetica", 11)
c.setFont("Courier", 10)
c.setFillColor(HexColor('#000000'))
c.drawCentredString(4 * inch, 0.2 * inch, timeM)

c.showPage()
c.setFont("Courier", 10)
c.setFillColor(HexColor('#000000'))
c.drawCentredString(4 * inch, 0.2 * inch, timeM)
f5 = Frame(1 * inch, 0.9 * inch, 6 * inch, 10 * inch, showBoundary=1)
f5.addFromList(story, c)
c.setFont("Helvetica", 23)
c.drawString(180, 740, "kube-apiserver.YAML")
c.drawImage(image, 2.2 * inch, 11 * inch, width=230, height=45)
c.drawImage(image2, 1.15 * inch, 6.1 * inch, width=400, height=275)


# c.setFillColor(HexColor('#000000'))



c.setFont("Helvetica", 18)
c.drawString(180, 420, "Passed")
c.setFillColor(HexColor('#038a27'))
c.drawString(200, 390, str(passed[1]))
```

```
c.setFillColor(HexColor('#000000'))
c.drawString(340, 420, "Failed")
c.setFillColor(HexColor('#000000'))
c.setFillColor(HexColor('#FF0000'))
c.drawString(357, 390, str(failed[1]))
c.setFillColor(HexColor('#000000'))
c.setFont("Helvetica", 11)







c.showPage()

c.setFont("Courier", 10)
c.setFillColor(HexColor('#000000'))
c.drawCentredString(4 * inch, 0.2 * inch, timeM)
f6 = Frame(1 * inch, 0.9 * inch, 6 * inch, 10 * inch, showBoundary=1)
f6.addFromList(story, c)
c.setFont("Helvetica", 23)
c.drawString(150, 740, "kube-controller-manager.YAML")
c.drawImage(image, 2.2 * inch, 11 * inch, width=230, height=45)
c.drawImage(image4, 1.15 * inch, 6.1 * inch, width=400, height=275)


# c.setFillColor(HexColor('#000000'))



c.setFont("Helvetica", 18)
c.drawString(180, 420, "Passed")
c.setFillColor(HexColor('#038a27'))
c.drawString(200, 390, str(passed[2]))

c.setFillColor(HexColor('#000000'))
c.drawString(340, 420, "Failed")
c.setFillColor(HexColor('#000000'))
c.setFillColor(HexColor('#FF0000'))
c.drawString(357, 390, str(failed[2]))
c.setFillColor(HexColor('#000000'))
c.setFont("Helvetica", 11)

c.showPage()
c.setFont("Courier", 10)
c.setFillColor(HexColor('#000000'))
c.drawCentredString(4 * inch, 0.2 * inch, timeM)
f7 = Frame(1 * inch, 0.9 * inch, 6 * inch, 10 * inch, showBoundary=1)
f7.addFromList(story, c)
c.setFont("Helvetica", 23)
c.drawString(230, 740, "etcd.YAML")
c.drawImage(image, 2.2 * inch, 11 * inch, width=230, height=45)
c.drawImage(image5, 1.15 * inch, 6.1 * inch, width=400, height=275)


# c.setFillColor(HexColor('#000000'))
```

```
c.setFont("Helvetica", 18)
c.drawString(180, 420, "Passed")
c.setFillColor(HexColor('#038a27'))
c.drawString(200, 390, str(passed[3]))

c.setFillColor(HexColor('#000000'))
c.drawString(340, 420, "Failed")
c.setFillColor(HexColor('#000000'))
c.setFillColor(HexColor('#FF0000'))
c.drawString(357, 390, str(failed[3]))
c.setFillColor(HexColor('#000000'))
c.setFont("Helvetica", 11)

c.setFillColor(HexColor('#038a27'))
c.setFontSize(12)
c.setFillColor(HexColor('#e62727'))
c.save()
```

ReportHTML.py

```
import json
import matplotlib.pyplot as plt
import numpy as np
import time
from time import time, ctime
json1 = "kube-scheduler.json", "kube-apiserver.json", "kube-controller-
manager.json", "etcd.json"

def Summary():

    passed = []
    failed = []

    for j in json1:
        with open(j, 'r') as json_file:
            json_load = json.load(json_file)

        passedJ = (json_load['summary']['passed'])
        passed.append(passedJ)



    for j in json1:
        with open(j, 'r') as json_file:
            json_load = json.load(json_file)

        failedJ = (json_load['summary']['failed'])
        failed.append(failedJ)

    labels = ['Scheduler', 'Apiserver', 'Controller-Manager', 'etcd']

    x = np.arange(len(labels))  # the label locations
    width = 0.35  # the width of the bars

    fig, ax = plt.subplots()
    rects1 = ax.bar(x - width / 2, passed, width, color="green",
label='Passed')
```

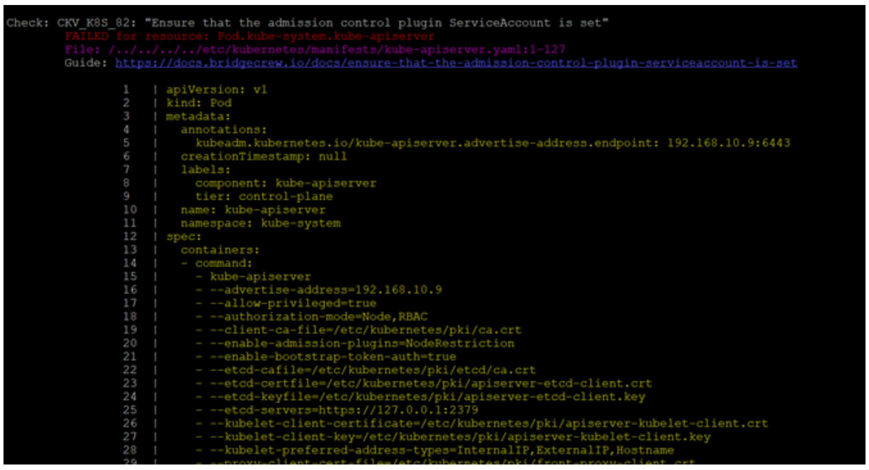
```
    rects2 = ax.bar(x + width / 2, failed, width, color="red",
label='Failed')

    ax.set_ylabel('Scores')
    ax.set_title('Scores of Kubernetes YAML files')
    ax.set_xticks(x, labels)
    ax.legend()

    ax.bar_label(rects1, padding=3)
    ax.bar_label(rects2, padding=3)

    fig.tight_layout()

    plt.savefig("Summary.png")

def sched():
    with open(json1[0], 'r') as json_file:
        json_load = json.load(json_file)
    y = []
    counterCapa = 0
    counterRootC = 0
    counterSeriveA = 0
    counterLimits = 0
    counterImage = 0
    counterTotal = 0
    MiscCalc = 0


    test = json_load["results"]["failed_checks"]

    for index in range(len(test)):
        for key, value in test[index].items():
                # print(value)
                # print(test)
                if key == "check_class":
                    # y = [value]
                    # print(y)
                    counterTotal = counterTotal + 1

                    if "Capabilities" in value:
                        counterCapa = counterCapa + 1

                    elif "RootContainers" in value:
                        counterRootC = counterRootC + 1

                    elif "Limits" in value:
                        counterLimits = counterLimits + 1

                    elif "Service" in value:
                        counterSeriveA = counterSeriveA + 1

                    elif "Image" in value:
                        counterImage = counterImage + 1


    Misc = counterCapa + counterRootC + counterSeriveA + counterImage +
counterLimits
    MiscCalc = counterTotal - Misc
    CounterT = [counterCapa, counterRootC, counterSeriveA, counterImage,
MiscCalc]
    labels = 'Capabilities', 'RootContainer', 'Limits', 'Service', 'Image'
    fig1, ax1 = plt.subplots()
    ax1.pie(CounterT, labels=labels, autopct='%1.1f%%',
            shadow=True, startangle=90)
    ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a
circle.
    ax1.set_title("Distribution of Failed Checks in kube-scheduler.yaml")
    plt.show()
    plt.savefig("kube-scheduler.png")


def API():
    with open(json1[1], 'r') as json_file:
        json_load = json.load(json_file)
    y = []
    counterServiceAcc = 0
    counterAuditLog = 0
    counterContainer = 0
    counterLimits = 0
    counterApi = 0
    counterTotal = 0
    MiscCalc = 0


    test = json_load["results"]["failed_checks"]

    for index in range(len(test)):
        for key, value in test[index].items():
                # print(value)
                if key == "check_class":
                    # y = [value]
                    # print(y)
                    counterTotal = counterTotal + 1
                    if "ServiceAccount" in value:
                        counterServiceAcc = counterServiceAcc + 1

                    elif "Container" in value:
                        counterContainer = counterContainer + 1

                    elif "AuditLog" in value:
                        counterAuditLog = counterAuditLog + 1

                    elif "Api" in value:
                        counterApi = counterApi + 1

                    elif "Limitis" in value:
                        counterLimits = counterLimits + 1



    Misc = counterApi + counterAuditLog + counterContainer +
counterServiceAcc + counterLimits
    MiscCalc = counterTotal - Misc

    CounterT = [counterServiceAcc, counterAuditLog, counterContainer,
counterApi, MiscCalc]

    labels = 'ServiceAccount', 'AuditLog', 'Container', 'API', 'Misc'

    fig1, ax1 = plt.subplots()
    ax1.pie(CounterT, labels=labels, autopct='%1.1f%%',
            shadow=True, startangle=90)
    ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a
circle.
    ax1.set_title("Distribution of Failed Checks in kube-apiserver.yaml")
    plt.show()
    plt.savefig("kube-apiserver.png")

def CM():
    with open(json1[2], 'r') as json_file:
        json_load = json.load(json_file)
    y = []
    counterCapa = 0
    counterRootC = 0
    counterSeriveA = 0
    counterLimits = 0
    counterImage = 0
    counterTotal = 0
    counterKCM = 0
    MiscCalc = 0


    test = json_load["results"]["failed_checks"]

    for index in range(len(test)):
        for key, value in test[index].items():
                # print(value)
                # print(test)
                if key == "check_class":
                    # y = [value]
                    # print(y)
                    counterTotal = counterTotal + 1

                    if "Capabilities" in value:
                        counterCapa = counterCapa + 1

                    elif "RootContainers" in value:
                        counterRootC = counterRootC + 1

                    elif "Limits" in value:
                        counterLimits = counterLimits + 1

                    elif "Service" in value:
                        counterSeriveA = counterSeriveA + 1

                    elif "Image" in value:
                        counterImage = counterImage + 1
                    elif "KubeControllerManager" in value:
                        counterKCM = counterKCM + 1




    Misc = counterCapa + counterRootC + counterSeriveA + counterImage +
counterLimits + counterKCM
    MiscCalc = counterTotal - Misc

    CounterT = [counterCapa, counterRootC, counterSeriveA, counterImage,
counterKCM, MiscCalc]
    labels = 'Capabilities', 'RootContainer', 'Service', 'Image', 'KCM',
'Misc'
```

```
    fig1, ax1 = plt.subplots()
    ax1.pie(CounterT, labels=labels, autopct='%1.1f%%',
            shadow=True, startangle=90)
    ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a
circle.
    ax1.set_title("Distribution of Failed Checks in kube-controller-
manager.yaml")
    plt.show()
    plt.savefig("kube-controller-manager.png")

def etcd():
    with open(json1[3], 'r') as json_file:
        json_load = json.load(json_file)
    y = []
    counterCapa = 0
    counterRootC = 0
    counterSeriveA = 0
    counterLimits = 0
    counterImage = 0
    counterTotal = 0
    counterKCM = 0
    MiscCalc = 0


    test = json_load["results"]["failed_checks"]

    for index in range(len(test)):
        for key, value in test[index].items():
                # print(value)
                # print(test)
                if key == "check_class":
                    # y = [value]
                    # print(y)
                    counterTotal = counterTotal + 1

                    if "Capabilities" in value:
                        counterCapa = counterCapa + 1

                    elif "RootContainers" in value:
                        counterRootC = counterRootC + 1

                    elif "Limits" in value:
                        counterLimits = counterLimits + 1

                    elif "Service" in value:
                        counterSeriveA = counterSeriveA + 1

                    elif "Image" in value:
                        counterImage = counterImage + 1
                    elif "KubeControllerManager" in value:
                        counterKCM = counterKCM + 1




    Misc = counterCapa + counterRootC + counterSeriveA + counterImage +
counterLimits
    MiscCalc = counterTotal - Misc
    CounterT = [counterCapa, counterRootC, counterSeriveA, counterImage,
counterLimits, MiscCalc]
    labels = 'Capabilities', 'RootContainer', 'Service', 'Image',
'Limits','Misc'

    fig1, ax1 = plt.subplots()
    ax1.pie(CounterT, labels=labels, autopct='%1.1f%%',
            shadow=True, startangle=90)
    ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a
circle.
    ax1.set_title("Distribution of Failed Checks in etcd.yaml")
    plt.show()
    plt.savefig("etcd.png")



def html():
    passed = []
    failed = []

    for j in json1:
        with open(j, 'r') as json_file:
            json_load = json.load(json_file)

        passedJ = (json_load['summary']['passed'])
        passed.append(passedJ)
    for j in json1:
        with open(j, 'r') as json_file:
            json_load = json.load(json_file)

        failedJ = (json_load['summary']['failed'])
        failed.append(failedJ)

    passedT = sum(passed)
    failedT = sum(failed)



    # print(failedT)
    # print(passedT)
    t = time()
    timeM = ctime(t)
    text = f"""

   <!DOCTYPE html>
<html lang="en">
<head>
<title>CSS Template</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="HTML.css">
</head>
<body>

<header>
  <h2>Kubernetes Hardening Report</h2>
</header>
<img id = "sum" src="Summary.png" alt="Trulli" width="600" height="370">
<h4 id = "hedp">Passed:</h4>
    <p class = "passed1">{passedT}</p>
    <h4 id ="hedf">Failed:</h4>
    <p class = "failed1">{failedT}</p>
    </div>
<section>
  <nav>
    <img id = "kube-scheduler" src="kube-scheduler.png" alt="Trulli"
width="374" height="300">
  </nav>

  <article>
    <h1 id = "Sched">kube-scheduler.yaml</h1>
    <h4>Passed:</h4>
    <p class = "passed">{passed[0]}</p>
    <h4>Failed:</h4>
    <p class = "failed">{failed[0]}</p>
    <p>The Kubernetes scheduler is a control plane process which assigns
Pods to Nodes. The scheduler determines which Nodes are valid placements
for each Pod in the scheduling queue according to constraints and available
resources. The scheduler then ranks each valid Node and binds the Pod to a
suitable Node</p>
  </article>
</section>

<section>
  <nav>
    <img id = "kube-scheduler" src="kube-apiserver.png" alt="Trulli"
width="374" height="300">
  </nav>

  <article>
    <h1 id = "Sched">kube-apiserver.YAML</h1>
    <h4>Passed:</h4>
    <p class = "passed">{passed[1]}</p>
    <h4>Failed:</h4>
    <p class = "failed">{failed[1]}</p>
    <p>The Kubernetes API server validates and configures data for the api
objects which include pods, services, replicationcontrollers, and others.
The API Server services REST operations and provides the frontend to the
cluster's shared state through which all other components interact.</p>
  </article>
</section>

<section>
  <nav>
    <img id = "kube-scheduler" src="kube-controller-manager.png"
alt="Trulli" width="374" height="300">
  </nav>

  <article>
    <h1 id = "Sched">kube-controller-manager.YAML</h1>
    <h4>Passed:</h4>
    <p class = "passed">{passed[2]}</p>
    <h4>Failed:</h4>
    <p class = "failed">{failed[2]}</p>
    <p>The Kubernetes controller manager is a daemon that embeds the core
control loops shipped with Kubernetes. In applications of robotics and
automation, a control loop is a non-terminating loop that regulates the
state of the system.</p>
  </article>
</section>

<section>
```

```
  <nav>
    <img id = "kube-scheduler" src="etcd.png" alt="Trulli" width="374"
height="300">
  </nav>

  <article>
    <h1 id = "Sched">etcd.YAML</h1>
    <h4>Passed:</h4>
    <p class = "passed">{passed[3]}</p>
    <h4>Failed:</h4>
    <p class = "failed">{failed[3]}</p>
    <p>etcd is a consistent and highly-available key value store used as
Kubernetes' backing store for all cluster data.</p>
  </article>
</section>
<footer>
<p>{timeM}</p>
</footer>

</body>

</html>


    """


    strSec = str(t)
    strSec = 'HTML_Report' + strSec + ".HTML"

    file = open(strSec, "w")

    file.write(text)

    file.close()





def main():
    Summary()
    sched()
    API()
    CM()
    etcd()
    html()



main()
```

Static Analysis Tools Commands & Output

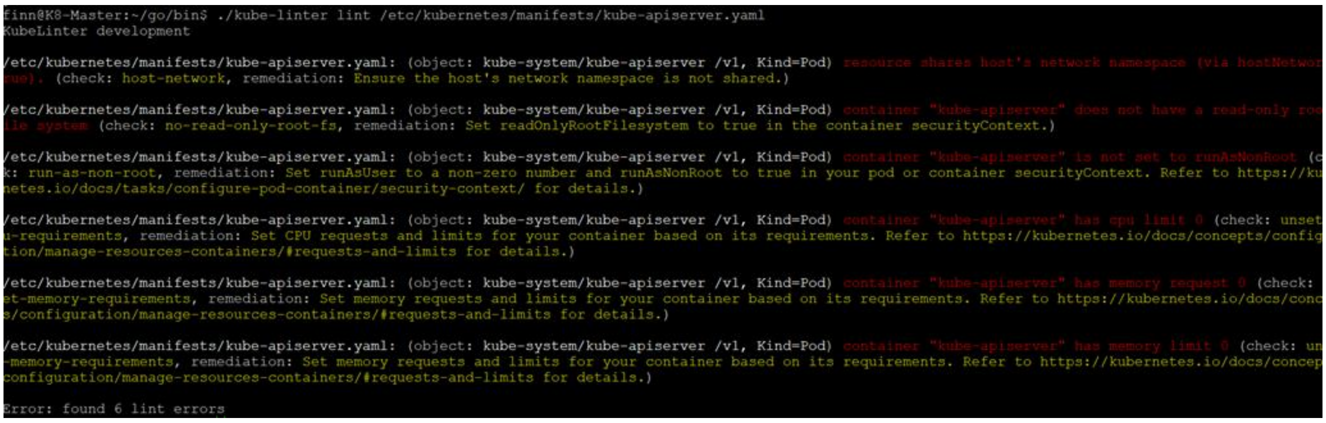
Checkov



```
Check: CKV_K8S_82: "Ensure that the admission control plugin ServiceAccount is set"
        FAILED for resource: Pod.kube-system.kube-apiserver
        File: /../../../../etc/kubernetes/manifests/kube-apiserver.yaml:1-127
        Guide: https://docs.bridgecrew.io/docs/ensure-that-the-admission-control-plugin-serviceaccount-is-set

                | apiVersion: v1
                | kind: Pod
                | metadata:
                |   annotations:
                |     kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 192.168.10.9:6443
                |   creationTimestamp: null
                |   labels:
                |     component: kube-apiserver
                |     tier: control-plane
                |   name: kube-apiserver
                |   namespace: kube-system
                | spec:
                |   containers:
                |   - command:
                |     - kube-apiserver
                |     - --advertise-address=192.168.10.9
                |     - --allow-privileged=true
                |     - --authorization-mode=Node,RBAC
                |     - --client-ca-file=/etc/kubernetes/pki/ca.crt
                |     - --enable-admission-plugins=NodeRestriction
                |     - --enable-bootstrap-token-auth=true
                |     - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
                |     - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
                |     - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
                |     - --etcd-servers=https://127.0.0.1:2379
                |     - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
                |     - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
                |     - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
                |     - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
```

Figure 22 - Checkov Output & Command

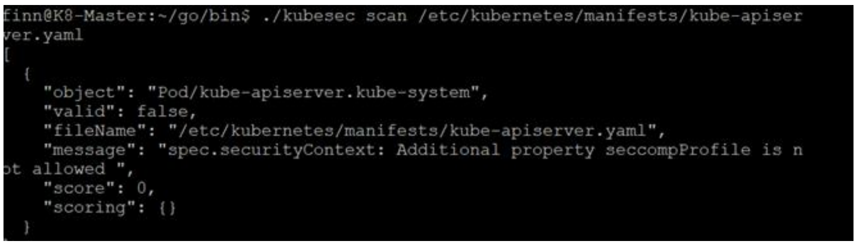
Kubelinter



```
finn@K8-Master:~/go/bin$ ./kube-linter lint /etc/kubernetes/manifests/kube-apiserver.yaml
KubeLinter development

/etc/kubernetes/manifests/kube-apiserver.yaml: (object: kube-system/kube-apiserver /v1, Kind=Pod) resource shares host's network namespace (via hostNetwor
rue). (check: host-network, remediation: Ensure the host's network namespace is not shared.)

/etc/kubernetes/manifests/kube-apiserver.yaml: (object: kube-system/kube-apiserver /v1, Kind=Pod) container "kube-apiserver" does not have a read-only roo
ile system (check: no-read-only-root-fs, remediation: Set readOnlyRootFilesystem to true in the container securityContext.)

/etc/kubernetes/manifests/kube-apiserver.yaml: (object: kube-system/kube-apiserver /v1, Kind=Pod) container "kube-apiserver" is not set to runAsNonRoot (c
k: run-as-non-root, remediation: Set runAsUser to a non-zero number and runAsNonRoot to true in your pod or container securityContext. Refer to https://ku
netes.io/docs/tasks/configure-pod-container/security-context/ for details.)

/etc/kubernetes/manifests/kube-apiserver.yaml: (object: kube-system/kube-apiserver /v1, Kind=Pod) container "kube-apiserver" has cpu limit 0 (check: unset
u-requirements, remediation: Set CPU requests and limits for your container based on its requirements. Refer to https://kubernetes.io/docs/concepts/config
tion/manage-resources-containers/#requests-and-limits for details.)

/etc/kubernetes/manifests/kube-apiserver.yaml: (object: kube-system/kube-apiserver /v1, Kind=Pod) container "kube-apiserver" has memory request 0 (check:
et-memory-requirements, remediation: Set memory requests and limits for your container based on its requirements. Refer to https://kubernetes.io/docs/conc
s/configuration/manage-resources-containers/#requests-and-limits for details.)

/etc/kubernetes/manifests/kube-apiserver.yaml: (object: kube-system/kube-apiserver /v1, Kind=Pod) container "kube-apiserver" has memory limit 0 (check: un
-memory-requirements, remediation: Set memory requests and limits for your container based on its requirements. Refer to https://kubernetes.io/docs/concep
configuration/manage-resources-containers/#requests-and-limits for details.)

Error: found 6 lint errors
```

Figure 23 - Kubelinter Command & Output

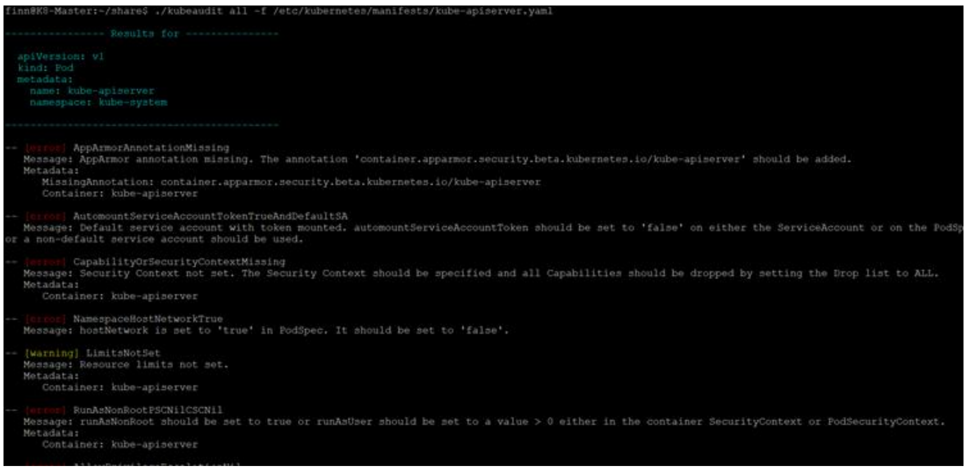
Kubesec



```
finn@K8-Master:~/go/bin$ ./kubesec scan /etc/kubernetes/manifests/kube-apiser
ver.yaml
[
  {
    "object": "Pod/kube-apiserver.kube-system",
    "valid": false,
    "fileName": "/etc/kubernetes/manifests/kube-apiserver.yaml",
    "message": "spec.securityContext: Additional property seccompProfile is n
ot allowed ",
    "score": 0,
    "scoring": {}
  }
```

Figure 24 – Kubesec Command & Ouput

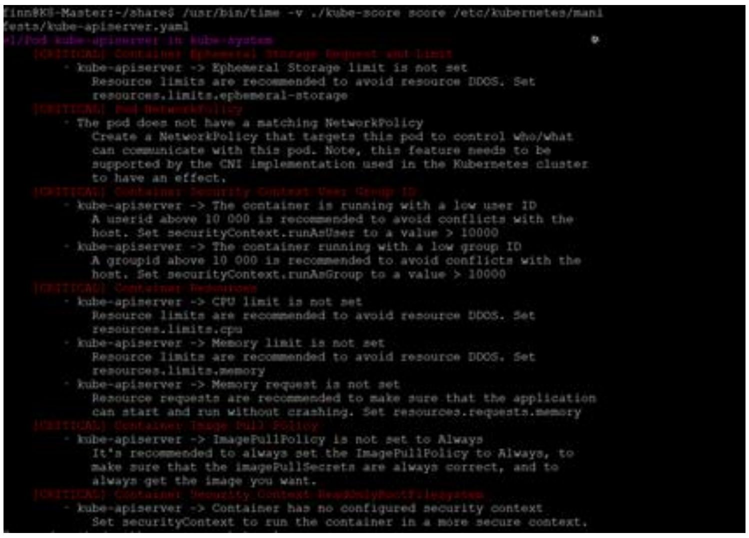
Kubeaudit



```
finn@K8-Master:~/share$ ./kubeaudit all -f /etc/kubernetes/manifests/kube-apiserver.yaml

---------------- Results for ---------------

  apiVersion: v1
  kind: Pod
  metadata:
    name: kube-apiserver
    namespace: kube-system

--------------------------------------------

-- [error] AppArmorAnnotationMissing
   Message: AppArmor annotation missing. The annotation 'container.apparmor.security.beta.kubernetes.io/kube-apiserver' should be added.
   Metadata:
      MissingAnnotation: container.apparmor.security.beta.kubernetes.io/kube-apiserver
      Container: kube-apiserver

-- [error] AutomountServiceAccountTokenTrueAndDefaultSA
   Message: Default service account with token mounted. automountServiceAccountToken should be set to 'false' on either the ServiceAccount or on the PodSp
or a non-default service account should be used.

-- [error] CapabilityOrSecurityContextMissing
   Message: Security Context not set. The Security Context should be specified and all Capabilities should be dropped by setting the Drop list to ALL.
   Metadata:
      Container: kube-apiserver

-- [error] NamespaceHostNetworkTrue
   Message: hostNetwork is set to 'true' in PodSpec. It should be set to 'false'.

-- [warning] LimitsNotSet
   Message: Resource limits not set.
   Metadata:
      Container: kube-apiserver

-- [error] RunAsNonRootPSCNilCSCNil
   Message: runAsNonRoot should be set to true or runAsUser should be set to a value > 0 either in the container SecurityContext or PodSecurityContext.
   Metadata:
      Container: kube-apiserver
```

Figure 25 - Kubeaudit Command & Output

Kubescore



```
finn@K8-Master:~/share$ /usr/bin/time -v ./kube-score score /etc/kubernetes/mani
fests/kube-apiserver.yaml
v1/Pod kube-apiserver in kube-system
    [CRITICAL] Container Ephemeral Storage Request and Limit
        · kube-apiserver -> Ephemeral Storage limit is not set
          Resource limits are recommended to avoid resource DDOS. Set
          resources.limits.ephemeral-storage
    [CRITICAL] Pod NetworkPolicy
        · The pod does not have a matching NetworkPolicy
          Create a NetworkPolicy that targets this pod to control who/what
          can communicate with this pod. Note, this feature needs to be
          supported by the CNI implementation used in the Kubernetes cluster
          to have an effect.
    [CRITICAL] Container Security Context User Group ID
        · kube-apiserver -> The container is running with a low user ID
          A userid above 10 000 is recommended to avoid conflicts with the
          host. Set securityContext.runAsUser to a value > 10000
        · kube-apiserver -> The container running with a low group ID
          A groupid above 10 000 is recommended to avoid conflicts with the
          host. Set securityContext.runAsGroup to a value > 10000
    [CRITICAL] Container Resources
        · kube-apiserver -> CPU limit is not set
          Resource limits are recommended to avoid resource DDOS. Set
          resources.limits.cpu
        · kube-apiserver -> Memory limit is not set
          Resource limits are recommended to avoid resource DDOS. Set
          resources.limits.memory
        · kube-apiserver -> Memory request is not set
          Resource requests are recommended to make sure that the application
          can start and run without crashing. Set resources.requests.memory
    [CRITICAL] Container Image Pull Policy
        · kube-apiserver -> ImagePullPolicy is not set to Always
          It's recommended to always set the ImagePullPolicy to Always, to
          make sure that the imagePullSecrets are always correct, and to
          always get the image you want.
    [CRITICAL] Container Security Context ReadOnlyRootFilesystem
        · kube-apiserver -> Container has no configured security context
          Set securityContext to run the container in a more secure context.
```

Figure 26 - Kubescore Command & Output