Hardening Kubernetes Clusters

Reducing Attack Surface in Kubernetes by means of Rootless Containers, Network Policies and Role Based Access Control

Bachelor Thesis

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science in Engineering

to the University of Applied Sciences FH Campus Wien

Bachelor Degree Program: Computer Science and Digital Communications

Author:

Guntram Björn Klaus

Student identification number:

c2110475170

Supervisor:

BSc. MSc. Bernhard Taufner

Date:

15.01.2024

Guntram Björn Klaus   *i*

Abstract

Kubernetes is a modern, extendable platform which abstracts the orchestration of containers. As institutions increasingly rely on Kubernetes in their production environments, especially those native in the cloud, securing the Kubernetes environment becomes paramount. This thesis analyzes CVEs to investigate Kubernetes from a security perspective. Solutions to minimize susceptibility to those problems found in prior collected CVEs are elaborated. Kubernetes configurations for API access, container runtimes and the container network that follow the concept of least privilege are presented as solutions.

Kurzfassung

Kubernetes ist eine moderne, erweiterbare Plattform, die die Orchestrierung von Containern abstrahiert. Da Institutionen in ihren Produktionsumgebungen zunehmend auf Kubernetes setzen, insbesondere jene die in der Cloud Software hosten, wird die Sicherung der Kubernetes-Umgebung immer wichtiger. Diese Arbeit analysiert CVEs, um Kubernetes aus einer Sicherheitsperspektive zu untersuchen. Lösungen zur Minimierung der Anfälligkeit für die Probleme, die in den zuvor gesammelten CVEs gefunden wurden, werden ausgearbeitet. Kubernetes-Konfigurationen für den API-Zugang, die Container-Laufzeiten und das Container-Netzwerk die dem Konzept der geringsten Privilegien folgen, werden als Lösungen vorgestellt.

List of Abbreviations

API	Application Programming Interface
CNI	Container Network Interface
CPU	Central Processing Unit
DNS	Domain Name System
IPC	Inter Process Communication
JSON	JavaScript Object Notation
JWT	JSON Web Token
NIST	National Institute Of Standards And Technology (USA)
OS	Operating System
POSIX	Portable Operating System Interface
PV	Persistent Volume
PVC	Persistent Volume Claim
RAM	Random Access Memory
REST	Representational State Transfer
UTS	Unix Time Sharing

Key Terms

Cloud

Container

Kubernetes

Security

Contents

1 Introduction

1.1 State Of The Art: Enterprises and Cloud

Over the last years, there has been a significant shift in the way enterprises around the world deploy, scale and maintain the lifecycle of their computing infrastructure and the software that runs on top of it. Applications used to be primarily constructed as one large software unit that bundled all features, business logic, user interfaces and data access components. Increasing necessity for scalability, flexibility and maintainability made organizations transition from such monolithic architectures to microservices, which are smaller, separated but loosely coupled software units implementing one component of the larger system at hand. The release of the Docker container platform in 2013 had a significant impact on making transitions from monoliths to microservices feasible. Kubernetes has since emerged as the go-to choice for managing containerized applications at scale, particularly in cloud-native environments. Its ability to automate tasks, scale applications, and support a wide range of use cases makes it a powerful tool for both large enterprises and small teams. However, Kubernetes is a large and complex system that consists of many components including additional plugins. To ensure an enterprise grade, production ready Kubernetes environment, there are vast amounts of configurations to make. As a single platform that abstracts orchestration, networking, storage, monitoring, access control and more, Kubernetes becomes open to attack from many different angles. The complexity, customizability and vastness highlight the amount of precision required to implement Kubernetes for production environments.

1.2 Related Work

The master thesis of Martínez at Uppsala University in 2023 studies security issues in certain Kubernetes architectures and discusses tradeoffs and opportunities. The study emphasizes tradeoffs of implementing different security solutions in terms of performance, deploy time and necessary overhead. [1]

At KTH Stockholm, Francesco Giangiulio and Sebastien Malmberg thoroughly tested the security posture of a production Kubernetes environment. The study was practically conducted on a specific company use case in Microsoft Azure. The thesis at hand differs in so far as no specific company use case is analyzed. Rather a general perspective on the matter is given. [2]

Nordell's master thesis offers a systematic evaluation of CVEs and provides experimental parts in which steps to practically reproduce exploits are presented. The author approaches the topic more systematically through an evaluation matrix. [3]

At JKU Linz, Hauber presents a concrete scenario of taking over Kubernetes clusters by automating attack chains. The extent of damage that can be made by exploiting misconfigurations is shown. The work at hand does not discuss specific scenarios, but collects and analyzes CVEs which would comprise possible scenarios. [4]

1.3 Objectives And Methodology

This thesis aims to investigate the state of Kubernetes in terms of security. What are known vulnerabilities in Kubernetes environments and is there a common denominator between them? Which concepts recur across the found vulnerabilities and incidents? To answer these questions, the literature review in chapter three explores the security threat landscape of Kubernetes environments. Specifically, recurring concepts and common denominators across vulnerabilities are identified and discussed. For this, Google, the Common Vulnerabilities and Exposures (CVE) database, the IEEE database, the ACM digital library as well as the official Kubernetes feed of CVEs are queried using keywords pertaining to Container and Kubernetes Security. In order to minimize manual labor to research the sheer amount of CVE reports, a Python script was developed to automate the research by querying public APIs. The script first queries the OpenCVE API for all reports containing keywords related to Kubernetes. With the collected CVE IDs, the official NIST API is called to find associated severity scores. All acquired papers, articles and slected CVE descriptions are skimmed through. The most relevant results are narrowed down and selected for closer inspection. Chapter four presents strategies to mitigate exploitability and impact of vulnerabilities found in chapter three.

2 Concepts

2.1 Containervirtualization

The high level definition of a container is that of a lightweight, standalone, and executable software package A container bundles the code itself, any required runtime environments, dependencies, settings as well as system tools and libraries into an isolated environment, which is instantiated on the underlying host machine. Such isolated environments are defined in tangible, portable formats called container images. The isolation of environments is mainly attributed to the namespace and Cgroup technology of the Linux kernel. [5]

2.1.1 Linux Kernel

The kernel: The operating system kernel is the core component of any operating system. It acts as a communication bridge between a user's applications and the underlying hardware. It is responsible for managing the fundamental functionality of the system, from scheduling processes to providing resources to applications. The kernel performs a number of crucial functions, including low-level networking, disk storage, and the control of processes and memory. The primary goals of the kernel include:

- Establish the process that will run next on the CPU

- Keep track of the amount of memory being used of each process

- Act as mediator between processes and hardware

- Handle requests for execution from the processes through system calls

The kernel performs its tasks in its own, designated part of the system memory called "kernel space". In contrast, custom applications such as web-browsers, texteditors and mail clients run in "user space". Kernel space and user space are kept apart. While the kernel has access to all of the memory, processes operating in the user space can only access a portion of it by calling the kernel interface. Through such system calls, a user-level program invokes services from the kernel. [6]

Linux Control Groups

Cgroups, short for control groups, are a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, etc.) of a collection of processes. These limits are enforced by the kernel. They ensure that processes do not consume excessive resources and that other processes can access the resources they need. Cgroups support hierarchical grouping of processes, allowing for flexible resource management based on application requirements and user-defined policies. Cgroups are used by Docker and container platforms to isolate and manage resource pool of containers. In a Kubernetes environment, the option to implement resource requests and limits of pods are enabled by Cgroups. [7]

Linux Namespaces

Namespaces are Linux kernel mechanisms that provide a hierarchical partitioning of the system environment. The Linux manpage defines namespaces as followed.

"A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource."

Linux provides seven main types of namespaces:

- PID namespace: Isolates process IDs

- Network namespace: Isolates Network devices, stacks, ports, etc.

- User namespaces: Isolates user and group IDs

- UTS namespace: Isolates hostname and domain name

- Mount namespaces: Isolates mount points

- IPC namespace: Isolates System V IPC objects and POSIX message queues

- Cgroup namespaces: Isolates Cgroup root directory

[8]

2.1.2 Container Images and Runtimes

Piecing together aforementioned concepts, companies like Docker and RedHat, and initiatives such as the Open Container Interface, build sophisticated products and standards around these kernel features. A container image is a snapshot of a filesystem that contains the application's code, runtime, and dependencies. It encapsulates the entire environment required to run the application. A container runtime is a software platform that manages the creation, execution, and lifecycle of containers based on an image specification. It provides the underlying infrastructure and mechanisms for running isolated container instances. Popular container runtimes include Docker Engine, Podman, and CRI-O. The relationship between images and runtimes is that images are the blueprints for containers, while runtimes are the engines that bring them to life. The runtime receives a container image, extracts the required components, mounts the image's filesystem, creates cgroups and namespaces to instantiate the container environment.

2.2 Kubernetes

Kubernetes is an open-source system for managing containerized workloads and services. It provides a portable, extensible, and scalable platform that automates the deployment, scaling, and management of applications [9].

2.2.1 Components

A Kubernetes cluster is separated into a control plane and data plane. The control plane groups the components responsible for the underlying system of Kubernetes itself. The data plane hosts the actual user containers. Machines that run control-plane services are referred

to as master-nodes, whereas machines that host user applications are referred to as worker-nodes. There are four core control-plane components which work together to manage the cluster and ensure that applications run smoothly. These components run as containers themselves inside the reserved 'kube-system' namespace. [10]

API Server

The API server is the entrance to the Kubernetes cluster. It provides a REST API that allows users and applications to interact with the cluster and manage its resources. The API server is responsible for handling requests from clients, validating them, and enforcing access control policies. It allows you to query the state of the cluster, as well as the configuration of all resources and the status of all pods. [11]

ETCD

Etcd is a distributed key-value store that is used to store the cluster's state. It is highly available and replicated across multiple nodes, ensuring that the cluster's state is always consistent. Etcd is used by all of the control plane components to store information about the cluster, such as the configuration of resources and the status of pods. [12]

Kube Scheduler

The scheduler is responsible for assigning pods to nodes in the cluster. It takes into account the resources available on each node, as well as the resource requirements of each pod, to ensure that pods are placed on nodes where they can run successfully. The scheduler also tries to spread pods across multiple nodes to improve availability and resilience. [13]

Controller Manager

The controller manager is a process that runs on each node in the cluster. It is responsible for monitoring the state of the cluster and taking corrective action when necessary. The controller manager watches for events, such as the creation or deletion of pods, and takes action to ensure that the cluster remains in a healthy state. For example, the controller manager can restart pods that are unhealthy or create new pods to meet demand. [14]

Kube DNS

In addition, Kube-DNS provides DNS resolution for services and pods within the cluster. It creates DNS records for services, allowing pods to reach services by name rather than IP address. This simplifies the process of accessing services and makes it more consistent with traditional application development without the need to map IP addresses to service names. It also acts as the authoritative DNS server for the cluster's default domain, which is 'cluster.local'. This ensures that DNS queries for names within the cluster are resolved correctly and consistently across all pods in the cluster. For example, if a service named 'frontend' inside the 'myapp' namespace exposes an application, it can be called from another application inside the cluster via the domain 'frontend.myapp.svc.cluster.local' or just 'frontend.myapp'. [15]

Kubelet

The Kubelet is an agent process that runs on each node in a Kubernetes cluster. Its primary responsibility is to manage the lifecycle of Kubernetes pods, which are the fundamental unit

of deployment in Kubernetes. Kubelet ensures that pods are running, healthy, and up-to-date with the desired state. It also interacts with the Container Runtime Interface (CRI) to run and manage containers, and it communicates with the Kubernetes API server to retrieve and update pod information. [16]

Kube Proxy

Next to the Kubelet, the Kube-proxy, is a network proxy that runs on each node in a Kubernetes cluster. It is responsible for maintaining network connectivity between pods and services. Services are abstractions that represent groups of pods that expose their endpoints to the outside world. Kube-proxy translates service definitions into network rules that direct traffic to the appropriate pods. This ensures that requests to a service are routed to the correct pods, even if the pods are dynamically provisioned or scaled. Figure 2.1 gives a high level overview over Kubernetes components and concepts. [17]
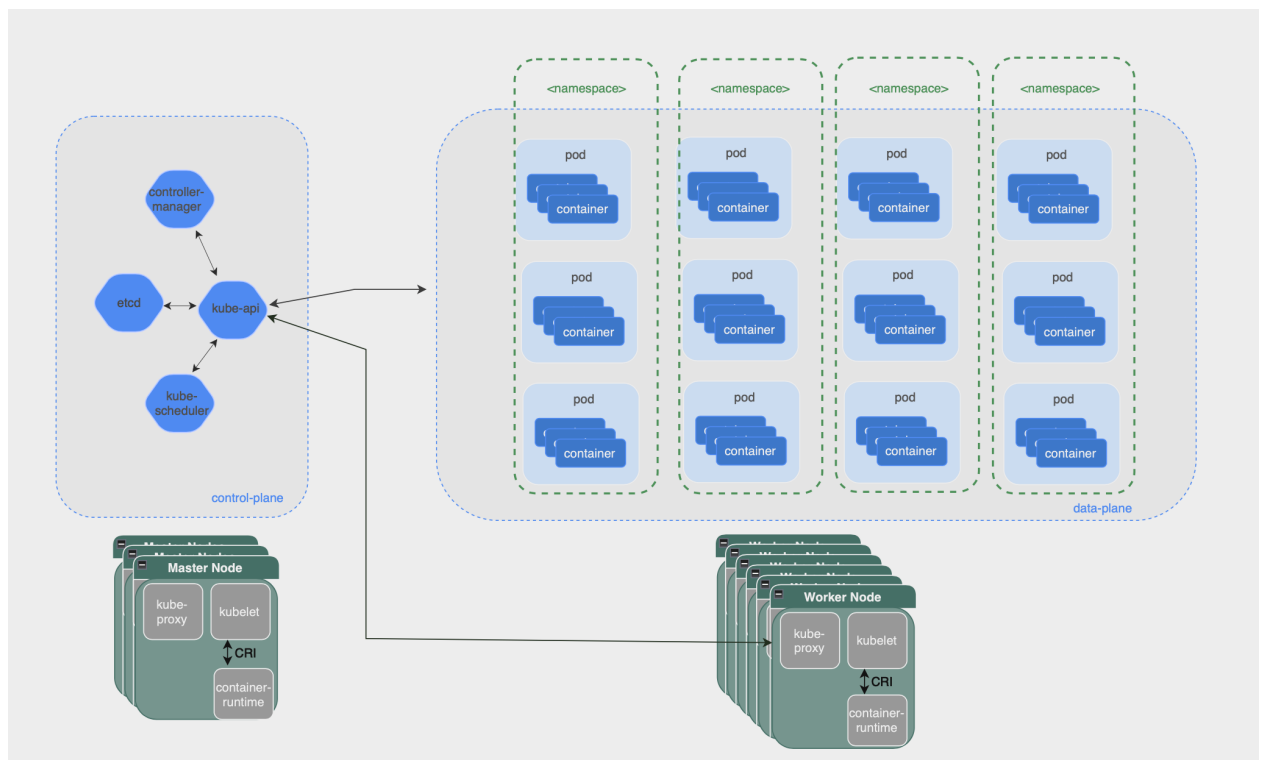

Figure 2.1: Kubernetes Architecture Diagram [source: author]

2.2.2 Core Cluster Objects

Kubernetes Objects are detailed specifications of the desired state of an application and the cluster as a whole. Objects are formulated in YAML or JSON syntax and are communicated to the Kubernetes API using the 'kubectl' command line utility [18].

Kubernetes Namespaces

Namespaces are virtual partitions in Kubernetes that arrange and segregate resource groupings inside a single cluster. Through namespaces the cluster is logically separated into more

manageable, smaller sections. Kubernetes objects which define the state of an application, such as 'Deployment', 'Service' and 'ConfigMap' are namespaced. Objects which define the cluster as a whole or which pertain to all applications in all namespaces are not namespaced and exist globally. Namespace isolation prevents unintended interactions or interference between applications, maintaining the integrity of each deployment. Namespaces can be assigned different security policies, allowing for the enforcement of custom authorization rules and access control mechanisms. This helps protect resources and restrict unauthorized access between selected namespaces within the cluster. [19]

Pod

In Kubernetes, pods are fundamental building blocks for deploying and managing containerized applications. A pod is a unit of one or more containers, which belong together and share resources such as network namespaces and filesystems. This tight coupling allows pods to be treated as a single unit for scheduling, management, and resource allocation. Pods expose the ports exposed by its containers, allowing external traffic to reach its intendend recipient. [20]

Deployment

Deployments describe the desired state of an application. A Deployment defines containers of pods, amount of pod replicas, hostnames, security settings, rolling update specifications and more. Its the central object for instantiating applications in Kubernetes. [21]

Service

In Kubernetes, a Service is an abstraction layer that defines a logical grouping of Pods and exposes them as a single, unified resource to the network. It simplifies the process of accessing and managing Pods by providing a stable endpoint and load balancing capabilities. Providing a stable endpoint ensures that pods can dynamically scale in the background with changing IP addresses. Services can be resolved to DNS names, providing a consistent and human-readable way to address the application. Service objects use selectors to identify the Pods that belong to the Service. [22]

ConfigMap

A ConfigMap is a key-value data store that can be mounted into Pods as environment variables, files, or command-line arguments. It provides a way to decouple configuration data from container images, making applications more portable and flexible. [23]

Ingress

In Kubernetes, an Ingress is a controller that manages external traffic routing to a set of Services within the cluster. It acts as a front door for applications, handling load balancing, SSL termination, and other advanced traffic management tasks. They direct external traffic to the appropriate services within the cluster. Ingresses support SSL/TLS termination, routing traffic based on the request's path, headers or query parameters, and more. [24]

ServiceAccount

A ServiceAccount is a security identity that allows pods to access Kubernetes resources and services. It acts as an authentication mechanism for pods, managing their access permissions

and controlling their interactions with the cluster. ServiceAccounts can mount Secrets into Pods, providing them with access to sensitive data such as passwords, API keys, or certificates. [25]

PersistentVolume and PersistentVolumeClaim

In Kubernetes, a PersistentVolume is a storage resource that provides persistent storage to Pods. It ensures that data remains accessible even after Pods are terminated and recreated. A PersistentVolumeClaim object is a request for persistent storage from a Kubernetes pod. It defines the desired storage characteristics and specifications, such as storage capacity and access modes. A PVC is referenced in a Deployment object. [26]

NetworkPolicy

A NetworkPolicy object defines rules for controlling network traffic between Pods. It allows for fine-grained control over network access, ensuring that Pods can only communicate with authorized sources and destinations. The rules are enforced by the Kubernetes network plugin, ensuring that only authorized traffic is allowed to pass between pods. [27]

3 Literature Review

3.1 Surveys

It is evident that the reliance on Kubernetes comes with the need of a clear security initiative. According to RedHat's report on the state of Kubernetes security of 2022, more than ninety percent of polled organizations underwent at least one security incident in their Kubernetes environment, which, in a third of cases, lead to the loss of revenue or customers. The majority of these incidents were detections of misconfigurations. About a third of respondents reported major vulnerabilities and runtime security incidents in relation to containers and/or Kubernetes which required immediate remediation. [28] In a more recent, similar report conducted by RedHat in 2023, two thirds of respondents had to delay or slow down application deployments because of security concerns. This is a significant increase compared to the 2022 survey, where just over half of participants experienced delays. Three of the most frequently mentioned advantages of containerization include quicker release cycles, quicker bug fixes, and increased flexibility to operate and manage applications. But if security is neglected, you can lose out on containerization's biggest benefit: agility. It becomes apparent that Kubernetes is not something that is installed once and then never looked at again. Rather, a container-based environement that leverages this orchestration technology requires attention for detail and constant, rigurous inspection, despite the great amount of abstraction provided and due to its highly customizable nature. [29]

3.2 CVE Scores

Kubernetes is highly customizable and offers a range of configuration options which determine the security posture of individual applications and the cluster as a whole. It shall be noted that Kubernetes vulnerabilities do not only entail standard Kubernetes components, but also add-ons deployed on top of 'plain' Kubernetes. Such can be the Nginx Ingress-Controller, a Service-Mesh, CI/CD tools closely embedded into Kubernetes and more. Generally, this can be anything that extends the Kubernetes API through Custom Resource Definitions (CRDs). CVE: The main goal of the Common Vulnerabilities and Exposures (CVE) Programme is to identify vulnerabilities in a unique way and link particular code bases (such as common libraries and software) to those vulnerabilities. The usage of CVEs guarantees that when discussing or exchanging information about a specific vulnerability, two or more parties can confidently refer to a CVE identifier (CVE numbers). The CVE program defines a vulnerability as:

"A weakness in the computational logic (e.g., code) found in software and hardware components that, when exploited, results in a negative impact to confidentiality, integrity, or availability. Mitigation of the vulnerabilities in this context typically involves coding changes, but could also include specification changes or even specification deprecations (e.g., removal of affected protocols or functionality in their entirety). [30]"

In addition, the Common Vulnerability Scoring System (CVSS) "is a method used to supply a qualitative measure of severity" [30]. The three metric groups that make up CVSS are Base, Temporal, and Environmental. Once the Temporal and Environmental metrics have

been scored, the Base metrics produce a score between 0 and 10. A vector string, which is a condensed textual representation of the values required to calculate the score, is another way to visualise a CVSS score. Therefore, enterprises, organisations, and governments that require precise and consistent vulnerability severity scores can benefit greatly from using CVSS as a standard measurement system. CVSS is frequently used to prioritise vulnerability mitigation efforts and to determine the severity of vulnerabilities found on a system. All publicly available CVE records are assessed by the National Vulnerability Database (NVD). There are multiple public APIs which allow for querying for various information on CVEs. The goal of the Python script is to quickly collect CVEs for a given keyword with a CVSS base score higher than a given integer. To achieve this, the OpenCVE project's API is queried for all CVEs containing a given keyword. The returned JSON response is filtered for the CVE number. This number is then used to call the NVD API of the NIST organization, whose response can then be filtered for the base severity score according to the CVSS framework. As a result, interesting insights can be gained by programmatic means. The following figure depicts the collected data on CVE's for the keyword 'Kubernetes'.

Total results: 269
Base score average: 7.13
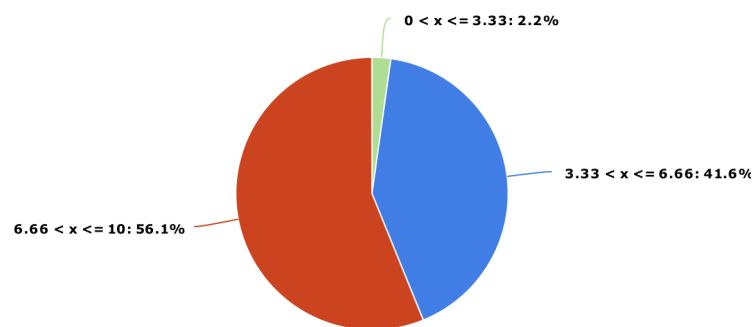Base score median: 7.2
Base score modus: 6.5

Figure 3.1: Makeup of base scores of CVEs about Kubernetes [source: author]

The chart below depicts the collected data on CVE's containing either of the keywords 'containerd', 'CRI-O' or 'Docker Container'.

Total results: 75
Base score average: 7.9
Base score median: 7.8
Base score modus: 9.8

Based on collected data, it is noticeable that CVSS base severity scores in the realm of Containervirtualization and Kubernetes tend toward extremes on the higher end. For the keyword 'Kubernetes', more than half of CVEs are rated with a base score higher than 6.66, while there are hardly any scores smaller than 3.33. For CVEs that draw on popular con-
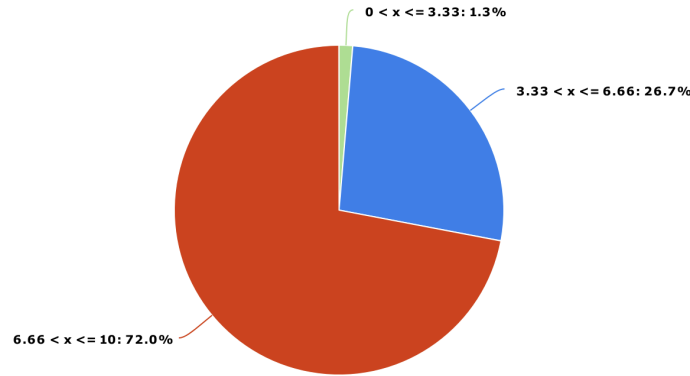
Figure 3.2: Makeup of base scores of CVEs about Kubernetes [source: author]

tainer runtimes, an overwhelming majority close to 75 percent of cases holds base scores between 6.66 and 10. Here, once again, scores smaller than 3.33 are low. These two observations additionally emphasize the necessity for clear security measures when orchestrating containers in Kubernetes. The script enables to quickly retrieve the tendencially more severe vulnerabilities. Some CVEs marked with a CVSS base scores higher than 7 shall be explored in the following chapters.

3.3 CVE Reports

Container Escape

Major vulnerabilities include those which fall under the category of a so called container-escape. Since a container is intended to be a runtime environment isolated from the underlying host, the concept of a container-escape relates to performing an exploit that breaks the confines of exactly this isolation, resulting in full or limited access to the underlying host machine or network. A study conducted by Reeves et al. at the end of 2021 investigates the susceptibility of different container runtime systems to escape-exploits by studying a batch of CVE reports. The study identifies three main causes for container escapes. [31]

CVE-2019-5736

First, mishandled file descriptors, if for example left accessible from within a container under /proc directory, provides malicious actors read and write access to the underlying host filesystem, as seen in CVE-2019-5736. In this reported vulnerability, a container is set up with a symlink from the container's entrypoint to proc/self/exe, which points back to its runC binary, which instantiated the container process. In addition, the container carries a harmful file which is designed to overwrite the file descriptors of any excecuting process that loads it. If an unknowing person executes a binary within the container, which has been manipulated to symlink to /proc/self/exe, the harmful file is able to overwrite the runC binary. The next time another, unrelated container is spawned, it is done by the compromised runC binary. [32]

CVE-2020-15257

Secondly, missing access control to runtime components could enable adversaries to gain access to UNIX sockets on the host, as reported in CVE-2020-15257. Here, it was possible to

connect to the containerd socket, thus enabling actors to issue API commands to freely create new containers on the host, unconstrained by Apparmor, Seccomp, or Linux capabilities. Thirdly, under 'adversary-controlled host execution' problems of similar fashion to mishandled file descriptors are mentioned. In this case however, vulnerability exposure starts with host binaries being executed in the container context, which makes it a target for manipulation. [33]

CVE-2019-101

In CVE-2019-101, the shared library "libc.so.6" is altered in such a way that it mounts the host filesystem when loaded. The new shell loads "libc.so" as soon as a developer runs "rkt-enter", which is the default /bin/bash command, to create a new shell. This sets off malicious code embedded in "libc.so", which uses the MKNOD syscall to construct a block device of the host root filesystem inside the container. As a result, the adversary is able to read and write to the host filesystem. [34] [35]

CVE-2022-0811

CVE-2022-0811, which is barely discussed in papers due to its young nature, reports a sophisticated container escape possibility of the CRI-O container engine. Generally, as explained in 2.1.1, the interface of the Linux kernel accepts parameters which control its behavior. This interface is consumed by CRI-O to set kernel options for a pod. However, the parameter input string is not checked or sanitized, which allows for injecting additional, otherwise undesired parameters. Specifically in the example of CVE-2022-0811, the "kernel.core_pattern" kernel parameter is specified within the parameter of the safe "kernel.shm_rmid_forced" parameter, which controls the kernel's reaction to a core dump. If a core dump is done in a CRI-O container, the parameter states the execution of a malicious binary. This binary sits inside the container but is invoked on the host in the root context of the container from the perspective of the kernel. [36] [37]

CVE-2022-23648

CVE-2022-23648 is another vulnerability related to container-escape, this time found in containerd, a popular Kubernetes runtime. The vulnerability is due to a vulnerable part of Containerd's CRI plugin that deals with volumes. It can be taken advantage of by an attacker by including a volume that contains path traversal in the image. This enables copying files from the host to a mounted path inside a container. More precisely, the "copyExisting-Contents" function in Containerd's code is where the vulnerability is located. This feature transfers the files from the volume path under the control of the attacker to a temporary folder that is subsequently mounted inside a container. Path traversal can be used by an attacker to deceive this method into copying any files from the host filesystem. This may result in private information being revealed. This can lead to the disclosure of confidential information. The severity of this vulnerability is rated as high, with a CVSS base score of 7.5. [38] [39]

CVE-2022-1708

CVE-2022-1708 is a vulnerability found in CRI-O, a lightweight container runtime for Kubernetes. This vulnerability is related to the allocation of resources without any limits or throttling, which can lead to uncontrolled resource consumption. The official CVE description of NIST states:

"The ExecSync request runs commands in a container and logs the output of the command. This output is then read by CRI-O after command execution, and it is read in a manner where the entire file corresponding to the output of the command is read in." [40]

It is thus possible to drain memory or disk space of the machine when CRI-O reads an extensive output. of the command. The vulnerability is rated as high, with a CVSS base score of 7.5 and targets system availability. [41]

CVE-2022-29179

Cilium is an eBPF-based CNI solution for network connectivity between workloads in Kubernetes Clusters. CVE-2022-29179 reports yet another subjection to the risk of container breakouts. The CVE number is not directly about a technique to gain access outside the container, it rather addresses a vulnerability in Cilium given a successful container escape. It explicitly mentions root containers as a prerequesite. According to this CVE entry, Cilium's service account in versions prior to version 1.9.16 allowed for escalating privileges to those of a cluster admin. This gave adversaries the ability to delete cluster resources such as pods and nodes. The entry is marked with a high base score of 8.2. [42]

CVE-2020-8554

In the case of CVE-2020-8554, the source is a design defect in the External IPs and Load Balancer IPs components of Kubernetes Services. An application running on a collection of pods can be exposed as a network service in an abstract manner using Kubernetes Services. One or more IPs are used to access a service. When the cluster's nodes are deployed, traffic intended for the service IPs will be routed through one of the backing pods that comprise the service. By assigning IP addresses that are already being used by other endpoints (internal or external), a malevolent user might intercept all cluster traffic directed towards those IP addresses. With the service shown below, it used to be possible to intercept UDP traffic to IP 8.8.8.8, which is Google's DNS server, and direct it to the "evil-dns-server" pod when it is deployed to the cluster. [43]

```yaml
! service.yaml
1    apiVersion: v1
2    kind: Service
3    metadata:
4      name: my-evil-service
5      namespace: my-evil-namespace
6    spec:
7      selector:
8        app: my-evil-dns-server
9      ports:
10       - name: dns
11         protocol: UDP
12         port: 53
13         targetPort: 9053
14     externalIPs:
15       - 8.8.8.8
16       - 8.8.4.4
```

Figure 3.3: YAML declaration of a malicious service [source: [44]]

CVE-2023-5043, CVE-2023-5044

Of the more recent vulnerabilities with high severity, CVE-2023-5043 and CVE-2023-5044 report problems with the Nginx Ingress Controller. The Nginx Ingress Controller is a popular

add-on to Kubernetes with which cluster incoming traffic is managed. Instead of assigning each Kubernetes Service an IP and port directly on the underlying node, the Nginx container serves as a single point of entry and acts as a reverse proxy to the cluster workloads. To expose a cluster workload, a Kubernetes resource of type 'Ingress' is defined. Within this YAML definition, the nginx ingress controller picks up custom configuration through so called annotation snippets. This allows for fine-grained, customized behavior of the controller for a respective service. It was however discovered that specific declarations of the "nginx.ingress.kubernetes.io/configuration-snippet" and "nginx.ingress.kubernetes.io/permanent-redirect" annotations of an Ingress Object can be used to inject arbitrary commands, making it possible to obtain credentials of the said nginx ingress controller. Utilizing this credential, even more cluster secrets could be obtained. Multi-tenant environments are most affected by the issue. [45] [46]

CVE-2022-24817

Flux2 is a continuous deployment tool which periodically aligns the state of a cluster with the state defined in a remote repository. As reported in CVE-2022-24817, when provided with a kubeconfig file with the correct access rights, such files can be used to define commands to be executed. [47] A perpatrator might create a kubeconfig file to run any code inside the controller's container if they had write access to a flux source or direct access to the target cluster. If the service account of the controller has high permissions, this can also result in privilege escalation in multi-tenancy setups. To exploit the issue, one of the following permission sets would need to be established within the range of affected versions:

- Direct access to the cluster for Kubernetes Secrets and Flux Kustomization

- Direct access to the cluster to change the Kubernetes secrets that are currently being utilised in Flux Kustomization.

- Direct access to the cluster allows for the creation or modification of existing Kubernetes secrets as well as Flux Kustomization.

- Authorization to alter a Flux Source's configuration. [48]

The vulnerability is marked with a CVSS base score of 9.9. [47]

Summary Of Findings

A successful container-escape poses one of the greatest risk in a Kubernetes environment as it provides a starting point from which all three pillars of the CIA triad can be targeted: Confidentiality, Integrity, and Availability. Once an attacker gains access of the host, secrets can be read, binaries can be altered, network traffic can be inspected and resources can be deleted. In contrast, other vulnerabilities 'only' allow for more restricted exploitability. For example, CVE-2022-1708 targets the availability of the system, but not the confidentiality or integrity of data. It can be seen that vulnerabilities vary tremendously in their appearance. CVE so and so targets a tool, CVE so and so targets the actual underlying kernel of a Kubernetes node. However, all these things can be prevented or alleviated by applying the concept of least privilege to containers, access to the Kubernetes control plane and the Kubernetes network. The foundation of any attack and its subsequent damage, whether they pertain to container instances or the network, is being able to freely move and operate from inside the cluster or a container.

4 Hardening Measures

Missing write permissions to sensitive directories are a major obstacle for performing container escapes. CVEs such as CVE-2019-5736 and CVE-2022-29179 become difficult to exploit. Missing permissions for accessing the Kubernetes API limit an attackers options to query for sensitive cluster data in case of compromise. Strict network settings prevent attackers from installing malicious software packages into the cluster. To harden clusters, the principle of least privilege must be applied. The principle of least privilege states that user or entity must only have access to data and permissions to perform actions required to complete their task. [49]

4.1 Rootless Containers

A rootless container is a container which runs its processes without root privileges. The root user in Linux is the account with the highest access rights on the instance. It can modify any system configuration, install and remove software packages, upgrade the operating system and firmware, read, write and execute any files on the system, and operate as any other user on the host. Any process that does not require such powerful access must be restricted in its permissions. The user which executes the application process inside the container must have the limited ability to perform only absolutely necessary actions to fulfill its purpose. [31]

Dockerfile

Running rootless containers in Kubernetes starts at the definition of the image. A Dockerfile is a text document which specifies commands to be executed for constructing the environment. This file is consumed by the Docker to build the image. The following Dockerfile, based on Alpine-Linux, installs nodejs, creates an unprivileged user with UID 1000, and assigns that user ownership to the /app directory, from where it can run its application process. [50]

```
1    FROM alpine:latest
2    RUN apk update && apk upgrade && apk add nodejs npm
3    RUN addgroup -g 1000 mygroup && adduser -u 1000 -D -G mygroup myuser
4    COPY --chown=1000:1000 ./server.js /app/
5    USER myuser
6    CMD ["node", "/app/server.js"]
```

Figure 4.1: Rootless image specification using Dockerfile [source: author]

Kubernetes offers the 'securityContext' configuration section for declaration of pods, deployments and statefulsets. Under 'securityContext', properties such as 'runAsNonRoot', 'runAsUser', 'allowPrivilegeEscalation' and 'readOnlyRootFilesystem' allow for restrictive access control settings inside containers. [51]

runAsNonRoot

This ensures that the container is started with a non-root user ID rather than the normal root UID of 0. Whenever possible, it is advised to run containers as non-root to limit the

possibilities to create privilege escalation or escape exploits. [51]

runAsUser

This setting in Kubernetes declares the user ID that will be used to run the container. [51]

allowPrivilegeEscalation

The ability to escalate a container's privileges is determined by this setting. When true, it gives a container access to extra privileges not available by default. When set to false, obtaining higher privileges, by 'sudo' or 'setuid' operations for example, will be denied. [51]

readOnlyRootFilesystem

When set to true, this property ensures that the root filesystem is not writable. [51]

Linux offers different security architectures such as AppArmor, SELinux, and Seccomp.

Capabilities

Capabilities are kernel level permissions that provide more precise control over kernel call permissions. Some examples of capabilities are the capacity to manage the network subsystem, modify file permissions, and carry out system-wide administrative tasks. Kubernetes offers configuration to remove or add capabilities using securityContext. Capabilities can be passed individually or as a comma-separated list in a string array. The -all shorthand is used to collectively drop or add all capabilitites. When the container runtime constructs the container, it uses this configuration to set capabilities. The container is given the default set of capabilities that the container runtime provides if the capabilities section is absent from the securityContext. [52]

AppArmor

AppArmor is a Linux kernel security module that allows administrators to restrict programs' capabilities with per-program profiles. Profiles can allow capabilities like network access, raw socket access, and the permission to read, write, or execute files on matching paths. AppArmor is supported in Kubernetes through the securityContext field. [53]

SELinux

Security-Enhanced Linux (SELinux) is a security module for Linux systems that provides a mechanism for supporting fine-grained access control security policies. Theconfiguration component of SELinux is a so called context label which defines permissions through combinations of user roles, types, levels, and security classes. The securityContext allows administrators to define such labels for Kubernetes deployments. The SELinux security module must be loaded on the host operating system to assing SELinux labels. [52]

Seccomp

The Linux kernel's seccomp feature can be used to restrict allowed system calls of a process, thereby lowering the kernel's attack surface. The set of system calls that the container is permitted to make is specified by seccomp-profiles also defined under securityContext. [54]

Thus, compromising the container instance through a reverse shell, for example, gives perpatrators limited ability to operate from inside the container, or break out of the container instance.

4.2 Network

A fresh installation of Kubernetes has no restrictions on network communication. Pods are free to receive ingress traffic from and send egress traffic to any other endpoints inside the cluster or the outside world. This overly permissive default setting must be restricted by applying 'NetworkPolicy' objects. Such objects require the installation of a Container Networking Interface (CNI) on top of Kubernetes. CNIs are Kubernetes plugins which set up the network environment of the cluster. It handles network interfaces, IP addresses and routing of traffic of containers. A CNI is a prerequesite for implementing Ingress, Service and NetworkPolicy objects. The NetworkPolicy rules work at Layer 3 (Network) and Layer 4 (Transport) of the OSI model. The following figure depicts a NetworkPolicy object which blocks all ingress and egress traffic. [55]

```
1    apiVersion: networking.k8s.io/v1
2    kind: NetworkPolicy
3    metadata:
4      name: block-all-traffic
5      namespace: my-app
6    spec:
7      podSelector: {}
8      policyTypes:
9      - Ingress
10     - Egress
```

Figure 4.2: A NetworkPolicy blocking all ingress and egress [source: [27]]

NetworkPolicies are additive. If there is a policy allowing traffic for certain IPs, ports and namespaces, it overrules a blocking policy. The following figure depicts a NetworkPolicy object which whitelists incoming traffic from the app-x namespace and outgoing traffic to the public IP of 91.213.77.62 on port 443.

```
1    apiVersion: networking.k8s.io/v1
2    kind: NetworkPolicy
3    metadata:
4     name: allow-app-x-and-egress-to-fh
5    spec:
6     podSelector: {}
7     policyTypes:
8     - Ingress
9     - Egress
10    ingress:
11    - from:
12     - namespaceSelector:
13         matchLabels:
14           name: app-x
15    egress:
16    - to:
17     - ipBlock:
18         cidr: 91.213.77.62/32
19     ports:
20     - protocol: TCP
21       port: 443
```

Figure 4.3: A NetworkPolicy allowing ingress from a namespace and egress to a public IP
        [source: author]

Standard NetworkPolicies are simple yet effective means to allow only the absolutely necessary origins and destinations of pod connections. More fine grained and powerful settings can be achieved through the implementation of a service-mesh. Istio and Linkerd, popular service-mesh solutions enable mutual TLS authentication in Kubernetes clusters. mTLS, an extension of standard TLS, requires both parties to authenticate themselves, instead of just the server authenticating itself to the client. In mTLS, both client and server present and verify certificates to establish a secure connection. mTLS is a crucial component for implementing zero trust networks. In addition to mTLS, service mesh solutions such as Linkerd and Istio provide AuthorizationPolicy objects. AuthorizationPolicies are used to granularly control access to specific endpoints of specific services with specific authorization protocols within the cluster. Figure 4.4 is an example of an Istio AuthorizationPolicy object, which implements the following rules for all pods inside the 'foo' namespace:

- Requests by pods identifying themselves with the cluster.local/ns/default/sa/sleep service account are allowed

- Request by pods inside the 'test' namespace are allowed

- GET methods to /info are only allowed when requests carry valid JWT issued by accounts.google.com

- POST methods to /data are only allowed when requests carry valid JWT issued by accounts.google.com

- Any other requests are denied

[56]

```
1    apiVersion: security.istio.io/v1beta1
2    kind: AuthorizationPolicy
3    metadata:
4      name: httpbin
5      namespace: foo
6    spec:
7      action: ALLOW
8      rules:
9      - from:
10       - source:
11           principals: ["cluster.local/ns/default/sa/sleep"]
12       - source:
13           namespaces: ["test"]
14        to:
15        - operation:
16            methods: ["GET"]
17            paths: ["/info*"]
18        - operation:
19            methods: ["POST"]
20            paths: ["/data"]
21        when:
22        - key: request.auth.claims[iss]
23          values: ["https://accounts.google.com"]
```

Figure 4.4: Granular access control with AuthorizationPolicy [source: [56]]

4.3 Role Based Access Control

RBAC is a authorization mechanism that controls which users or groups can access resources and perform specific actions within the cluster. It provides a fine-grained way to manage access permissions and enforce security policies across the Kubernetes environment.

Role And ClusterRole

Role objects define a set of permissions that can be granted to users or groups. They specify which resources can be accessed, what actions can be performed on those resources, and the scope of the permissions (namespace-wide or cluster-wide). ClusterRole objects are similar to Roles, but they have a broader scope, applying to all namespaces within the cluster. They are typically used to define permissions for system components or users who need access to resources across multiple namespaces. [57]

RoleBindings And ClusterRoleBinding

RoleBindings and ClusterRoleBindings associate Roles or ClusterRoles with users or groups, granting them the specified permissions. They define the mapping between identities and the permissions they hold. [57]

Users And Groups

Users represent individual identities in the Kubernetes system. They can be created manually or integrated with existing authentication providers, such as LDAP or Active Directory. Groups are collections of users that can be granted permissions together. This simplifies access management for related users and promotes role-based access control. [57]

```
1    api Version: rbac.authorization.k8s.io/v]
2    kind: ClusterRole
3    metadata:
4      name: pod-reader-role
5    rules:
6    - apiGroups: [""]
7      resources: ["pods"]
8      verbs: ["get", "watch", "list"]
```

Figure 4.5: Read access to pods in Kubernetes [source: [58]]

Figure 4.5 shows a role with minimal cluster access. A user or serviceaccount bound to this ClusterRole through a ClusterRoleBinding will have read access to all pods in the cluster. [58]

5 Discussion And Future Work

The amount of CVE listings related to containers and orchestration technically allow for more in depth analysis. There is still much more to be investigated and there are more improvements to be made to ensure a production grade Kubernetes environment. Exploring all CVE reports in depth would require more time and resources and would go beyond the scope of this bachelor thesis. Using Python, consuming APIs for CVE information greatly alleviated the overhead to understand the big picture of CVE reports. Future work could describe a more elaborate setup of Kubernetes including policy engines, external secret management, OIDC servers, image scanning and runtime observability tools.

Admission to the cluster can be controlled and validate using solutions like the Open Policy Agent or Kyverno. Using admission controllers, objects that do not adhere to custom policies are denied application to the cluster. This extends possibilities to enforce rootless containers, as the admission controller could block the installation of root deployments in the first place. [59]

Another aspect to consider in order to improve the security posture of the environment is secret management. General secrets, like database credentials and API keys, as well as SSL certificates and SSL keys used by applications deployed in Kubernetes, should be externalized from the source code and repository platforms like GitHub [60] Major cloud providers like AWS and Azure provide services for secret management, or self-hosted open source tools like Hashicorp Vault can be leveraged.

Service mesh implementations and self-hosted OIDC servers like Keycloak or cloud-provided identity management can be tightly integrated to ensure the most granular access control to applications and resources.

Additionally, image scanning in Kubernetes involves checking images used in your Kubernetes cluster for known weaknesses and suspicious installations or configurations. Trivy is an open-source scanner which works by examining the filesystem of the image and looking for known vulnerabilities in the installed software packages, libraries, and binaries. It uses multiple vulnerability databases to provide accurate and timely vulnerability reports. [61]

Runtime security solutions allow for monitoring and blocking certain kernel events or suspicious system calls. Tools like Falco, Tetragon and NeuVector help gain visibility into abnormal behavior of hosts and containers. [62]

6 Conclusion

This thesis aims to shine light on Kubernetes from a security perspective. With the help of scripting, data on CVE reports relating to containers and Kubernetes is collected and analyzed. From this data, significant cases are extracted and further explained in detail. Container and Kubernetes configurations are formulated to minimize attack surface. Rootless processes, strict network segmentation and minimal RBAC rules must be made to ensure a hardened environment. The thesis highlights the importance of a clear security initiative when utilizing Kubernetes in a production environment.

Bibliography

[1] A. R. Martinez, "Study of security issues in kubernetes (k8s) architectures; tradeoffs and opportunities," [Accessed 20.11.2023]. [Online]. Available: https://uu.diva-portal.org/smash/get/diva2:1781490/FULLTEXT01.pdf 1

[2] F. Giangiulio and S. Malmberg, "Testing the security of a kubernetes cluster in a production environment," [Accessed 20.11.2023]. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1700029/FULLTEXT01.pdf 1

[3] F. Nordell, "A systematic evaluation of cves and mitigation strategies for a kubernetes stack," [Accessed 20.11.2023]. [Online]. Available: https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=9103043&fileOId=9103371 1

[4] C. Hauber, "Taking over a kubernetes cluster: Automating an attack chain," [Accessed 20.11.2023]. [Online]. Available: https://ssw.jku.at/Teaching/BachelorTheses/2022/Hauber_Carina.pdf 1

[5] "Use containers to build, share and run your applications," [Accessed 10.11.2023]. [Online]. Available: https://www.docker.com/resources/what-container/ 3

[6] "What is the linux kernel?" February 2019, [Accessed 20.11.2023]. [Online]. Available: https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel 3

[7] "cgroups(7) — linux manual page," [Accessed 19.12.2023]. [Online]. Available: https://man7.org/linux/man-pages/man7/cgroups.7.html 3

[8] "namespaces (7) — linux manual page," [Accessed 19.12.2023]. [Online]. Available: https://man7.org/linux/man-pages/man7/namespaces.7.html 4

[9] "Kubernetes overview," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/overview/ 4

[10] "Kubernetes components," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/ 5

[11] "The kubernetes api," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/overview/kubernetes-api/ 5

[12] "Operating etcd clusters for kubernetes," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/ 5

[13] "Kubernetes scheduler," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/ 5

[14] "kube-controller-manager," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/ 5

[15] "Dns for services and pods," [Accessed 12.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/ 5

[16] "kubelet," [Accessed 12.11.2023]. [Online]. Available: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/ 6

[17] "kube-proxy," [Accessed 12.11.2023]. [Online]. Available: https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/ 6

[18] "Objects in kubernetes," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/ 6

[19] "Namespaces," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/ 7

[20] "Pods," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods/ 7

[21] "Deployments," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ 7

[22] "Service," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/service/ 7

[23] "Configmaps," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/configmap/ 7

[24] "Ingress," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/ingress/ 7

[25] "Service accounts," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/security/service-accounts/ 8

[26] "Persistent volumes," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/storage/persistent-volumes/ 8

[27] "Network policies," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/network-policies/ 8, 17, 26

[28] A. Kohgadai, "The state of kubernetes security in 2022," RedHat, Tech. Rep., 2022. 9

[29] ——, "The state of kubernetes security in 2023," RedHat, Tech. Rep., 2023. 9

[30] "Vulnerabilities," [Accessed 29.11.2023]. [Online]. Available: https://nvd.nist.gov/vuln 9

[31] Z. Guo, Z. Lv, N. Li, T. Yuan, X. Gao, and Z. Yuan, "Comprehensive defense scheme against container escape related to container management procedure," in *2022 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, 2022, pp. 263–266. 11, 15

[32] "CVE-2019-5736." Available from MITRE, CVE-ID CVE-2019-5736., 2019. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-5736 11

[33] "CVE-2020-15257." Available from MITRE, CVE-ID CVE-2020-15257., 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-15257 12

[34] M. Reeves, "Investigating escape vulnerabilities in container runtimes," Master's thesis, Purdue University, Indiana, 2021. 12

[35] "CVE-2019-101." Available from MITRE, CVE-ID CVE-2019-101., 2019. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-101 12

[36] "CVE-2022-0811." Available from MITRE, CVE-ID CVE-2022-0811., 2022. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-0811 12

[37] J. Walker, "cr8escape: New vulnerability in cri-o container engine discovered by crowdstrike (cve-2022-0811)," March 2022, [Accessed 16.12.2023]. [Online]. Available: https://www.crowdstrike.com/blog/cr8escape-new-vulnerability-discovered-in-cri-o-container-engine-cve-2022-0811/ 12

[38] M. Ahuje, "Cve-2022-23648: Kubernetes container escape using containerd cri plugin and mitigation," May 2022, [Accessed 22.11.2023]. [Online]. Available: https://www.crowdstrike.com/blog/understanding-cve-2022-23648-kubernetes-vulnerability/ 12

[39] "CVE-2022-23648." Available from MITRE, CVE-ID CVE-2022-23648., 2022. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-23648 12

[40] "Cve-2022-1708 detail," [Accessed 29.11.2023]. [Online]. Available: https://nvd.nist.gov/vuln/detail/cve-2022-1708 13

[41] "CVE-2022-1708." Available from MITRE, CVE-ID CVE-2022-1708., 2022. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-1708 13

[42] "CVE-2022-29179." Available from MITRE, CVE-ID CVE-2022-29179., 2022. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-29179 13

[43] "CVE-2020-8554." Available from MITRE, CVE-ID CVE-2020-8554., 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-8554 13

[44] "Cve-2020-8554: Man in the middle using loadbalancer or externalips," [Accessed 29.11.2023]. [Online]. Available: https://github.com/kubernetes/kubernetes/issues/97076 13, 26

[45] "CVE-2023-5043." Available from MITRE, CVE-ID CVE-2023-5043., 2023. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-5043 14

[46] "CVE-2023-5044." Available from MITRE, CVE-ID CVE-2023-5044., 2023. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-5044 14

[47] "CVE-2022-24817." Available from MITRE, CVE-ID CVE-2022-24817., 2023. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-24817 14

[48] "Improper kubeconfig validation allows arbitrary code execution," [Accessed 29.11.2023]. [Online]. Available: https://github.com/fluxcd/flux2/security/advisories/GHSA-vvmq-fwmg-2gjc 14

[49] "What is the principle of least privilege?" [Accessed 25.11.2023]. [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-the-principle-of-least-privilege 15

[50] "Overview of best practices for writing dockerfiles," [Accessed 25.12.2023]. [Online]. Available: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/ 15

[51] "Configure a security context for a pod or container," [Accessed 26.12.2023]. [Online]. Available: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/ 15, 16

[52] E. Smalling and M. Jarvis, "10 kubernetes security context settings you should understand," March 2021, [Accessed 16.12.2023]. [Online]. Available: https://snyk.io/blog/10-kubernetes-security-context-settings-you-should-understand/ 16

[53] "Restrict a container's access to resources with apparmor," [Accessed 26.12.2023]. [Online]. Available: https://kubernetes.io/docs/tutorials/security/apparmor/ 16

[54] "Restrict a container's syscalls with seccomp," [Accessed 26.12.2023]. [Online]. Available: https://kubernetes.io/docs/tutorials/security/seccomp/ 16

[55] "Network plugins," [Accessed 10.11.2023]. [Online]. Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/ 17

[56] "Authorization policy," [Accessed 29.12.2023]. [Online]. Available: https://istio.io/latest/docs/reference/config/security/authorization-policy/ 18, 26

[57] "Using rbac authorization," [Accessed 29.12.2023]. [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/rbac/ 19

[58] G. Rostami, "Role-based access control (rbac) authorization in kubernetes," *Journal of ICT Standardization*, vol. 11, no. 3, pp. 237–260, 2023. 19, 26

[59] "Require runasnonroot," [Accessed 28.12.2023]. [Online]. Available: https://kyverno.io/policies/pod-security/restricted/require-run-as-nonroot/require-run-as-nonroot/ 20

[60] J. Walsh, "Kubernetes security: best practices for kubernetes secrets management," September 2023, [Accessed 28.12.2023]. [Online]. Available: https://www.cncf.io/blog/2023/09/28/kubernetes-security-best-practices-for-kubernetes-secrets-management/ 20

[61] F. Campo, H. Wang, and J. Coffman, "Exploring solutions for container image security," in *2023 IEEE 14th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, 2023, pp. 82–88. 20

[62] K. German and O. Ponomareva, "An overview of container security in a kubernetes cluster," in *2023 IEEE Ural-Siberian Conference on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*, 2023, pp. 283–285. 20

List of Figures

Appendix

cve.py, retrieves CVE numbers and associated scores for a search term

```python
import requests
from requests.auth import HTTPBasicAuth
import time
from dotenv import load_dotenv
import os
import json

load_dotenv()
nvd_nist_api_key = os.getenv('NVDAPIKEY')
username = os.getenv('OPENCVEUSER')
password = os.getenv('OPENCVEPW')
headers = {"apiKey":f"{nvd_nist_api_key}"}

opencve_api_url = "https://www.opencve.io/api/cve?search=Kubernetes&page="
nvd_nist_api_url = "https://services.nvd.nist.gov/rest/json/cves/2.0?cveId="
page=1

cve_ids = []
cve_scores = []
start=0
end=19

while True:
    try:
        response = requests.get(opencve_api_url + str(page), auth=HTTPBasicAuth(username, password))

        if response.status_code == 404:
            print("Received a 404 response. Stopping the loop.")
            break

        if response.status_code == 200:
            data = response.json()
            for entry in data:
              cve_ids.append(entry["id"])

            print (cve_ids)

            for i in range(start, len(cve_ids)):
              time.sleep(1) #because rate limit
              print("Fetching data for: " + cve_ids[i])
              res = requests.get(nvd_nist_api_url + cve_ids[i], headers=headers )
              if res.status_code == 200:
                  print("Success")
                  data = res.json()
                  if "cvssMetricV31" in data["vulnerabilities"][0]["cve"]["metrics"]:
                      cve_scores.append(data["vulnerabilities"][0]["cve"]["metrics"]["cvssMetricV31"][0]["cvssData"]["baseScore"])
                  elif "cvssMetricV30" in data["vulnerabilities"][0]["cve"]["metrics"]:
                      cve_scores.append(data["vulnerabilities"][0]["cve"]["metrics"]["cvssMetricV30"][0]["cvssData"]["baseScore"])
                  elif "cvssMetricV2" in data["vulnerabilities"][0]["cve"]["metrics"]:
```

```
50                        cve_scores.append(data["vulnerabilities"][0]["cve"]["metrics"]["cvssMetricV2"][0]["cvssData"]["baseScore"])
51                    else:
52                        cve_scores.append(0)
53                else:
54                    cve_scores.append(0)
55                    print(f"Request failed with status code: {res.status_code}")
56            print("After page: "+str(page))
57            print(len(cve_ids))
58            print(len(cve_scores))
59            print("We have collected "+str(len(cve_ids))+ " CVEs")
60            page += 1
61            start += 20 #results are returned in batches of size 20
62            end += 20
63
64        else:
65            print(f"Request failed with status code: {response.status_code}")
66
67    except requests.exceptions.RequestException as e:
68        print(f"Request error: {e}")
69
70  print("Length of our arrays")
71  print(len(cve_ids))
72  print(len(cve_scores))
73
74  data_dict = {cve_ids[i]: cve_scores[i] for i in range(len(cve_ids))}
75  with open("output_k8.json", "w") as json_file:
76      json.dump(data_dict, json_file)
77
78  #We now have 2 arrays: cve_ids and cve_scores, whereby cve_ids[x] has a CVE base score of cve_scores[x]
79  #We filter all CVE's with a score > 7:
80
81  index=0
82  filtered_scores=[]
83  filtered_cves=[]
84  for score in cve_scores:
85      if score >= 7:
86          filtered_scores.append(score)
87          filtered_cves.append(cve_ids[index])
88      index += 1
89
90  print("All CVE's with a baseScore higher or equal than 7")
91  print(filtered_scores)
92  print(filtered_cves)
```