FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

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:

dd.mm.yyyy

Declaration of authorship:

I declare that this Bachelor Thesis has been written by myself. I have not used any other than the listed sources, nor have I received any unauthorized help.

I hereby certify that I have not submitted this Bachelor Thesis in any form (to a reviewer for assessment) either in Austria or abroad.

Furthermore, I assure that the (printed and electronic) copies I have submitted are identical.

Date: Signature:

Abstract

(E.g. “This thesis investigates...”)

Kurzfassung

(Z.B. “Diese Arbeit untersucht...”)

List of Abbreviations

ARP	Address Resolution Protocol
GPRS	General Packet Radio Service
GSM	Global System for Mobile communication
WLAN	Wireless Local Area Network

Key Terms

GSM

Mobilfunk

Zugriffsverfahren

Contents

1 Introduction

1.1 Background: Enterprises and Cloud

1.2 Research Objectives

1.3 Methodology

1.4 Structure

2 Concepts

2.1 Containervirtualization

2.1.1 Linux Kernel

2.1.2 Container Images

2.1.3 Container Runtimes

2.2 Kubernetes

2.2.1 Components

2.2.2 Cluster Architecture

2.2.3 Cluster Objects

3 Literature Review

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. The purpose of this literature review is to explore the security threat landscape of Kubernetes environments. Specifically, recurring concepts and common denominators across vulnerabilities shall be identified and discussed. For this, the database of Common Vulnerabilities and Exposures (CVE), the IEEE database, the ACM digital library and 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. The acquired papers, articles and CVE descriptions are skimmed through. The most relevant results are narrowed down and selected for closer inspection. 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)."
In addition, the Common Vulnerability Scoring System (CVSS) "is a method used to supply a qualitative measure of severity"CITE HERE. 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 figures

shall depict data on CVE's for the keyword 'Kubernetes'.

Pie chart Average value Median value modus value

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. 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. CITE REDHAT 2022 2023

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 and/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. 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. 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. In CVE-2019-101(44–47), 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" when the administrator runs "rkt-enter", which is the /bin/bash command by default, to create a new shell in the container. 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.

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, 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.
CVE-2022-23648 is another vulnerability related to container-escape, this time found in Containerd, a popular Kubernetes runtime. This vulnerability lies in Containerd's CRI plugin that handles OCI image specs containing "Volumes". An attacker can exploit this vulnerability by adding a Volume containing path traversal to the image. This allows them to copy arbitrary files from the host to a container mounted path. More specifically, The vulnerability resides in the "copyExistingContents" function in Containerd's code. This function copies the files from the attacker-controlled volume path to a temporary folder that is later mounted inside a container. An attacker can trick this function into copying arbitrary files from the host filesystem using path traversal. 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.
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 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." It is thus possible to exhaust memory or disk space of the node 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.
A successful container-escape arguably 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, vulnerabilities such as CVE-2022-1708 'only' target the availability of the system. The average CVE scores of
The foundation of such attacks is being able to either freely instantiate containers or freely move and operate from inside a container The most notable ones, which have a severity score above 8, according to the Common Vulnerability
We see 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 k8s node. However, all these things can be prevented or alleviated by applying the concept of least privilege to containers, access to Kubernetes and the Kubernetes network. For example, if a malicious actor cannot freely create files in any desired directory of a compromised container, it significantly reduces his ability to exploit a given vulnerability. Not being able to create that file in the first place, due to missing write permissions, would be a major obstacle for performing a container escape. It is not always possible to completely avoid a vulnerability. This is due to software bugs and unidentified weaknesses in the code that have passed through the testing and review process.

3.1 State of the Art

3.2 CVE Numbers

3.3 Incident Reports

4 Hardening Measures

4.1 Securing Containers

4.1.1 Linux File Permissions

4.1.2 Root versus Rootless

4.1.3 Rootless Policy Enforcement

4.2 Securing the Network

Default pod-to-pod network settings, as an example, allow open communication to quickly get a cluster up and running, at the expense of security hardening. Network segmentation.....

4.2.1 Network Policies

4.2.2 Service Mesh

4.3 Authentication and Authorization

4.3.1 Role Based Access Control

4.3.2 Kubeconfig Keys and Certificates

4.3.3 Service Accounts

5 Discussion

5.1 Implications for businesses

5.2 Tradeoffs and Difficulties

5.3 Complexity

6 Conclusion

7 Outlook / Future work

Bibliography

List of Figures

List of Tables

Appendix

(Hier können Schaltpläne, Programme usw. eingefügt werden.)