Universitat Oberta de Catalunya

Master's Thesis

Securing Kubernetes in Public Cloud Environments

Author: Víctor Martínez Bevià
Master's Degree: Cybersecurity and Privacy
Tutor: Manuel Jesús Mendoza Flores

To Encarna Maria, whose support knows no end.

Abstract

With the rise of cloud providers, it is now easier than ever to create a startup and pay for infrastructure "as you go" instead of having to invest in physical servers and storage. At the same time, Kubernetes provides a scalable platform that meshes perfectly with the elasticity of the cloud environment. The low entry fee coupled with the conveniences of the providers shouldering infrastructure costs due to the shared responsibility model makes companies jump at the opportunity and run their code with a sometimes questionable security posture.

In this work, we take a look at the current landscape of cybersecurity threats for Kubernetes clusters in a cloud environment, reviewing existing recommendations, best practices, and threat models in order to provide a structured guide on how to improve the security of the infrastructure against known attack vectors. Finally, we offer actionable implementations of each of the chosen security mitigations.

Contents

Chapter 1

Introduction

1.1 Motivation

Cybersecurity attacks, such as ransomware, data breaches, or phishing, are part of our daily lives as "cyber-crime is growing exponentially [and] the cost of cybercrime is predicted to hit $8 trillion in 2023 and will grow to $10.5 trillion by 2025"[1]. But even though "businesses [...] operate in a world in which 95% of cybersecurity issues can be traced to human error"[2], nowadays, "the primary hurdle companies have recently cited is a belief that the current cybersecurity posture is 'good enough'.[...] the notion of 'good enough' indicates a lack of specific metrics around measuring cybersecurity efforts."[3].

To shed light on cybersecurity attacks, many security-related companies conduct yearly surveys to gauge the state of the Cloud Native Security field. According to these surveys, as of 2022, "93% of respondents experienced at least one security incident in their Kubernetes environments in the last 12 months, sometimes leading to revenue or customer loss"[4] (*The State of Cloud Security Report 2022* by Snyk points to an 80%[5]). Additionally, "90 % of organizations cannot detect, contain, and resolve cyber threats within an hour"[6] (with Snyk reporting 89% of organizations[5]). A common struggle seems to be that "77% of organizations cite problems with poor training and collaboration as a major challenge"[5], while "only 10% consider their developers and security teams to be experts"[7].

Many people turn to existing Best Practices documents and CIS benchmarks due to a lack of expertise. However, these resources often present a list of generic mitigations that may not be useful for specific use cases. Every environment has unique requirements, and a one-size-fits-all checklist is unlikely to address all scenarios effectively. Instead, it is crucial to adopt a threat-model-based approach that tailors mitigations and best practices to the current situation. By defining the domain for each mitigation, we can select an implementation that suits our current environment. This approach will ensure that our security measures are effective and aligned with the specific risks and challenges we face.

1.2 Goals

This work aims to provide a set of functional security implementations following a threat-model-based approach which, if put in place and maintained, should mitigate the most common attacks for a Kubernetes cluster in a Cloud environment.

1.3 Planning

In order to achieve the objective of this thesis, it is necessary to undertake the following tasks.

- **Provide an attacks landscape for Kubernetes clusters**. The goal is to gather a list of agnostic attacks and tactics, so we can systematically identify vulnerable points. In order to help with the following chapters we also will:
  - Gather statistics of Kubernetes cyberattacks to later help with prioritization
  - Review existing Kubernetes security guides and benchmarks
  - Review existing Kubernetes threat models
- **Map attacks to mitigations**. Identify what we want to work on and why. To that end, we will:
  - Present a list of mitigations that counter our chosen attack vectors
  - Research possible solutions for the chosen mitigations
- **Implement mitigations**. Once the solutions have been chosen, provide an example of how to implement them in a Kubernetes cluster residing in the Cloud.
  - Decide which attack vectors apply to our current domain and which are out of scope
  - Categorize mitigations based on their domain of application
  - Key aspects to adopting a solution will be how cost-effective it is in terms of work, money, and frequency of attack

1.4 Risks

As there is no such thing as perfect security, so we must try to avoid unnecessary efforts due to wrong prioritization. It is necessary to follow a structured guide to apply mitigations because time and effort are limited and need to be taken into account. In Chapter 4, the study case will be presented, and the goal is to analyze the scenario and systematically apply only the necessary mitigations for our current environment.

Chapter 2

State of the Art

2.1 Anatomy of current Kubernetes Cybersecurity Attacks

It is a common practice for security providers to gather data by means of an annual survey. These surveys do not focus solely on Kubernetes, but on a wide range of topics, and the annual frequency allows them to keep up with the latest security trends. Nonetheless, they should be taken with a grain of salt, because they represent a relatively low sample, mostly from already knowledgeable companies.

In this work though, we will focus on the parts of those surveys that deal with Kubernetes security incidents. We have some surveys, like the ones from RedHat and PaloAlto, that focus on *attack vectors*:

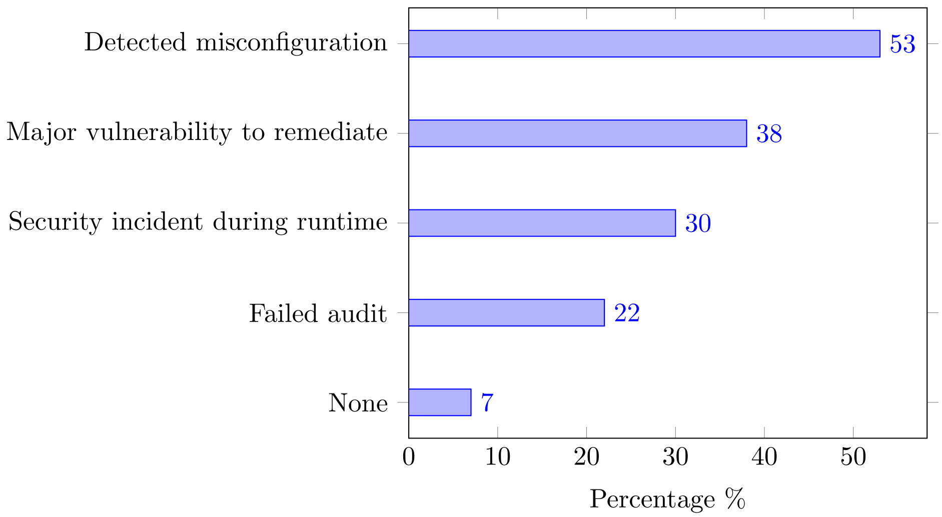

Figure 2.1: RedHat: "In the past 12 months, what security incidents or issues related to containers and/or Kubernetes have you experienced? (pick as many as apply)[4]"

Top 5 Security Incidents

1. Risk introduced early in application development
2. Workload images with vulnerabilities or malware
3. Vulnerable web applications and APIs
4. Unrestricted network access between workloads
5. Downtime due to misconfiguration

Figure 2.2: Palo Alto: "Top 5 Security Incidents [6]"

Others focus on the *attack goal*:

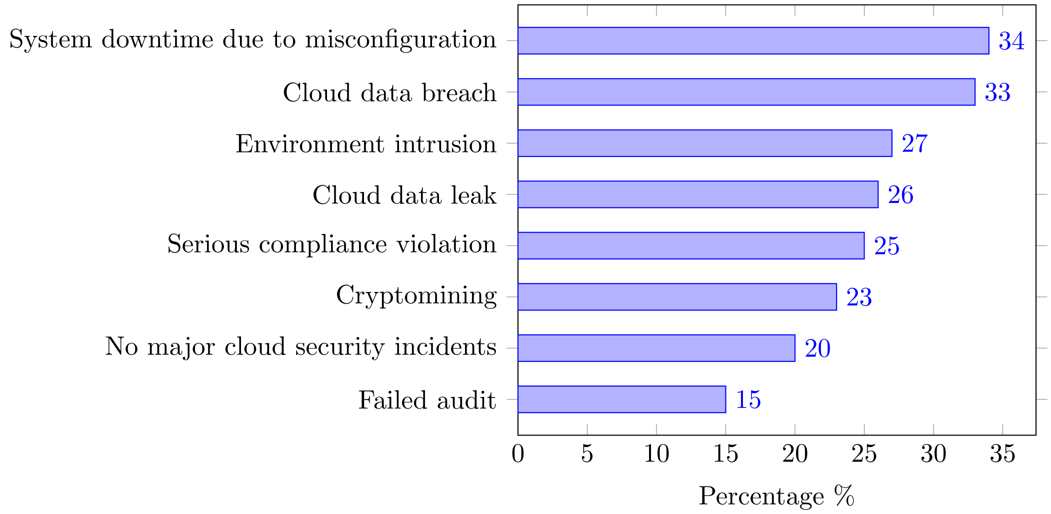

Figure 2.3: Snyk: "Serious cloud security incidents experienced[5]"

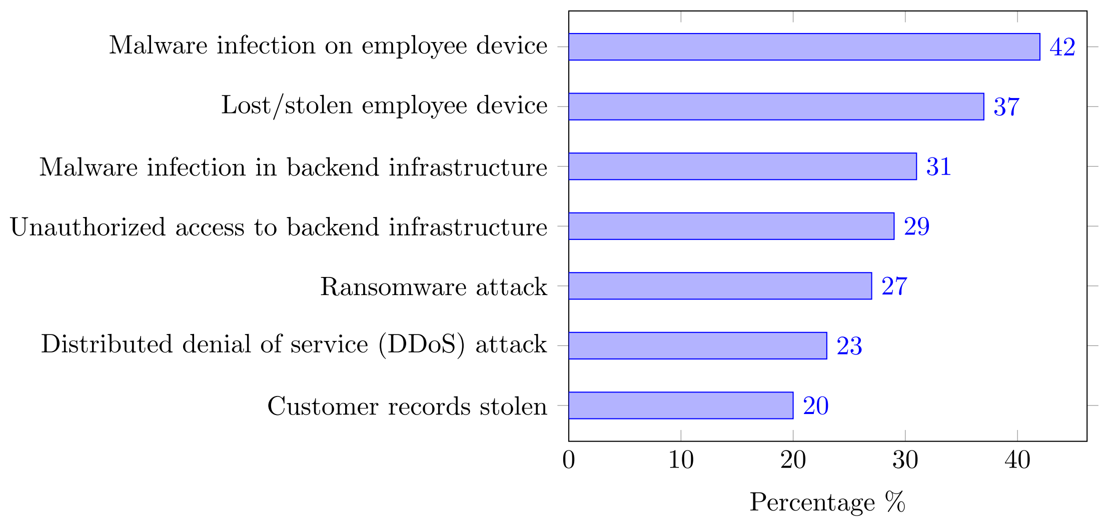

Figure 2.4: CompTIA: "Cybersecurity Incidents from Past Year[3]"

As these surveys do not follow a common template, the non-standard types of attacks make it challenging to collate data. Furthermore, the *attack vector* versus *goal* focus complicates the extraction of insights.

2.2 Kubernetes Security Standards

To help with security compliance, one can refer to the available *benchmarks*, *hardening guides* or *best practices* references. These come in the form of checklists or a group of more general actions to improve the security of the Kubernetes environments, and can help from the inexperienced operator deploying its initial cluster to the auditor assessing the security of an environment.

Several benchmark and hardening guides exist, with some of the most renowned being:

- The *Center for Internet Security (CIS) Kubernetes Benchmark* "is the product of a community consensus process and consists of secure configuration guidelines developed for Kubernetes"[9]. The Center for Internet Security is a nonprofit that provides a plethora of benchmarks, controls, and hardened images to help IT professionals safeguard against threats.
- The National Security Agency (NSA) and the Cyber Security and Infrastructure Security Agency (CISA) provide a *Cybersecurity Technical Report* called *Kubernetes Hardening guide*, developed "in furtherance of their respective cybersecurity missions, including their responsibilities to develop and issue cybersecurity specifications and mitigations."[10]
- Defense Information Systems Agency (DISA), part of the United States of America Department of Defense, has their own *Kubernetes Security Technical Implementation Guides (DISA STIGs)*. Reportedly it "is published as a tool to improve the security of Department of Defense (DoD) information systems. The requirements are derived from the National Institute of Standards and Technology (NIST) 800-53 and related documents"[11].
- The major Cloud providers also have their own hardening guides with specifics for their managed Kubernetes services:
  - Best Practices Guide for Security (Amazon Web Services Elastic Kubernetes Service)[12]
  - Harden your cluster's security (Google Kubernetes Engine) [13]
  - Best practices for cluster security and upgrades in Azure Kubernetes Service (AKS) [14]

It's important to note that being compliant does not mean being secure. Though these guides contain good information and generally provide accurate, actionable advice to improve your security, the problem comes when one of these guides is used as a one-and-only stop for achieving a secure environment. More often than not, they are incomplete, meaning they cannot cover your whole environment (nor is their purpose), so if you follow just one of them you could end up with a very secure Kubernetes deployment, but a cloud environment full of vulnerabilities. Or vice-versa. As Anais Ulrichs and Rory McCune from Aqua Security point out, "security Standards are helpful but they should be taken as something you use as a starting point"[15].

2.3 Threat Models

2.3.1 Sig-Security K8s Threat Model

In January 2020, the CNCF Financial User Group released a Kubernetes Threat Model[16]. This was an analysis of threats and mitigations following the STRIDE methodology. As it happens with the Benchmarks, the work "only focused on the Kubernetes platform itself, not on the full end-to-end container solution that would include the SDLC or additional applications used to monitor Kubernetes. These components and the wider environment are likely to be individual to a specific end-user". The threat model follows two approaches:

- The *Bottom-up Approach*, which "shows entry points throughout the Kubernetes platform with the aim of satisfying the stated goal." Among the goals are: *Denial of Service*, *Malicious Code Execution* and *Establish Persistence*.
- The *Scenario Approach*, "identifying attack vectors open to an attacker in certain scenarios". The two scenarios are: *Compromised application leads to foothold in container* and *Attacker on the network*.

These approaches' analysis highlights the following *Main Attack Vectors*:

- Service Token
- Compromised container
- Network endpoints
- Denial of Service
- RBAC Issues

Since 2020 there has not been another release of the threat model by the CNCF.

2.3.2 Microsoft Threat Matrix for Kubernetes

In April 2020, Microsoft published a post in their Security blog proposing a *Threat matrix for Kubernetes*. In that post, Microsoft "created the first Kubernetes attack matrix: an ATT&CK-like matrix comprising the major techniques that are relevant to container orchestration security, with focus on Kubernetes"[17]. A new version of the matrix was published a year later[18], which included new techniques and discontinued the outdated ones.

The *Microsoft Threat Matrix for Kubernetes*[19] classify different techniques utilized to target a Kubernetes cluster. These groups are referred to by *tactics*, and are essentially a subset of those listed in the MITRE ATT&CK Matrix for Enterprise[20]. The following tactics are included:

- Initial Access

- Execution
- Persistence
- Privilege Escalation
- Defense Evasion
- Credential Access
- Discovery
- Lateral Movement
- Collection
- Impact

Each of these tactics is comprised of several techniques which are not mutually exclusive. For example, exploiting *Exposed sensitive interfaces* serves both the *Initial Access* and *Discovery* tactics. It's worth mentioning that while Microsoft associates their techniques with those in the ATT&CK Matrix, they may not necessarily match up with the same tactics. Using the previous example, in MITRE the technique is called *External Remote Services*, but in this case, it is classified under *Initial Access* and *Persistence*.

2.3.3 Expanded Microsoft Threat Matrix for Kubernetes

The *Hacking Kubernetes* book by O'Reilly[21] introduces a community-expanded version of Microsoft's Threat Matrix. Additionally to adding techniques, it also replaces the tactic *Collection* for *Command & Control*. The expanded threat matrix look like this:

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access Kubernetes API server	Access cloud resources	Images from a private registry	Data destruction
Compromised image in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Collecting data from pod	Resource hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from proxy server	Application credentials in configuration files	Exposed sensitive interfaces	Application credentials in configuration files		Node scheduling DoS[1]
Exposed sensitive interfaces	SSH server running inside container	Container service account	Node to cluster escalation[1]	DNS tunneling/ exfiltration[1]	Access managed identity credentials	Instance Metadata API	Writable hostPath mount		Service Discovery DoS[1]
Compromise user endpoint[1]	Sidecar injection	Static pods	Control plane to cloud escalation[1]	Shadow admission control or API server[1]	Malicious admission controller	Compromise K8s Operator[1]	CoreDNS poisoning		PII or IP exfiltration[1]
Compromised host[1]	Container lifecycle hooks[1]	Sidecar injection[1]	Compromise admission controller[1]			Access host filesystem[1]	ARP poisoning and IP spoofing		Container pull rate limit DoS[1]
Compromised etcd[1]		Rewrite container lifecycle hooks, liveness[1]	Compromise K8s Operator[1]				Access K8s Operator[1]		SOC/SIEM DoS[1]
		K3d botnet[1]	Container breakout[1]						

[1] Community-expanded

Table 2.1: Microsoft Threat Matrix for Kubernetes (expanded)

2.4 Conclussions

We've seen that *Benchmarks* and *Hardening* guides are not sufficient by themselves as they lack the context of the current environment. Instead, they should be used as a reference to implement mitigations that apply to each case. By combining the use of a threat model with the information the current guides offer we can get a more structured security plan, as we will know why are we applying each mitigation.

In Chapter 3 we'll analyze an existing threat model, and review possible mitigations by using the mentioned guides in section 2.2 as well as other solutions where Kubernetes configurations are not enough.

Chapter 3

Preventive Controls

3.1 Mitigations

In the previous chapter, we took a look at the Kubernetes threat models available and as the focus of this thesis is to provide an entry point to secure a Kubernetes cluster, we will be following the Microsoft Threat Matrix for Kubernetes. The reason for the choice is that the model is widely available, it has an established reputation that makes the community build upon it, and it is presented in a structured manner that links each mitigation to existing MITRE mitigations.

As the mitigations can block more than one technique, we'll begin by gathering the common mitigations and reviewing possible solutions to implement in Chapter 4.

3.1.1 Mapping tactics to mitigations

Initial Access

The initial phase or step in a cyber attack is where an attacker gains unauthorized access to a target system or network. The initial foothold can be achieved via the cluster management layer or by finding and exploiting vulnerabilities in a Kubernetes container, enabling first access to the cluster.

Techniques	Mitigations
Using cloud credentials	**MS-M9001**: Multi-factor Authentication **MS-M9002**: Restrict access to the API server using IP firewall **MS-M9003**: Adhere to least-privilege principle
Compromised credentials	**MS-M9004**: Secure CI&CD environment **MS-M9005**: Image Assurance Policy
Kubeconfig file	**MS-M9003**: Adhere to least-privilege principle **MS-M9002**: Restrict access to the API server using IP firewall **MS-M9006**: Enable JIT elevated access to API server to limit attack surface or impact
Application vulnerability	**MS-M9005**: Image Assurance Policy **MS-M9007**: Network Intrusion Prevention
Exposed sensitive interfaces	**MS-M9008**: Limit Access to Services Over Network **MS-M9009**: Require Strong Authentication to Services **MS-M9014**: Network Segmentation
Compromise user endpoint	**MS-M9001**: Multi-factor Authentication

Table 3.1: Initial Access Tactic

Execution

Upon gaining access, the attackers employ different techniques to run their code inside a cluster.

Techniques	Mitigations
Exec into container	**MS-M9003**: Adhere to least-privilege principle **MS-M9010**: Restrict Exec Commands on Pods **MS-M9011**: Restrict Container runtime using LSM
Bash or cmd inside container	**MS-M9011**: Restrict Container runtime using LSM **MS-M9012**: Remove Tools from Container Images
New container	**MS-M9003**: Adhere to least-privilege principle **MS-M9013**: Restrict over permissive containers **MS-M9005.003**: Gate images deployed to Kubernetes cluster
Application exploit (RCE)	**MS-M9005**: Image Assurance Policy **MS-M9014**: Network Segmentation **MS-M9011**: Restrict Container runtime using LSM
SSH server running inside container	**MS-M9015**: Avoid Running Management interface on Containers **MS-M9014**: Network Segmentation **MS-M9011** Restrict Container runtime using LSM
Sidecar injection	**MS-M9003**: Adhere to least-privilege principle **MS-M9013**: Restrict over permissive containers **MS-M9005.003**: Gate images deployed to Kubernetes cluster
Container lifecycle hooks	**MS-M9003**: Adhere to least-privilege principle

Table 3.2: Execution Tactic

Persistence

One option for the attacker is to find a way to keep access to the cluster in case their initial foothold is lost.

Techniques	Mitigations
Backdoor container	**MS-M9003**: Adhere to least-privilege principle **MS-M9013**: Restrict over permissive containers **MS-M9005.003**: Gate images deployed to Kubernetes cluster
Writable hostPath mount	**MS-M9013**: Restrict over permissive containers **MS-M9016**: Restrict File and Directory Permissions **MS-M9011**: Restrict Container runtime using LSM **MS-M9017**: Ensure that pods meet defined Pod Security Standards
Kubernetes CronJob	**MS-M9005.003**: Gate images deployed to Kubernetes cluster **MS-M9003**: Adhere to least-privilege principle **MS-M9013**: Restrict over permissive containers
Malicious admission controller	**MS-M9003**: Adhere to least-privilege principle
Container service account	**MS-M9025**: Disable Service Account Auto Mount **MS-M9003**: Adhere to least-privilege principle
Static pods	**MS-M9016**: Restrict File and Directory Permissions **MS-M9032**: Avoid using web-hosted manifest for Kubelet
Sidecar injection	**MS-M9003**: Adhere to least-privilege principle **MS-M9013**: Restrict over permissive containers **MS-M9005.003**: Gate images deployed to Kubernetes cluster
Rewrite container lifecycle hooks, liveness	**MS-M9003**: Adhere to least-privilege principle

Table 3.3: Persistence Tactic

Privilege Escalation

The privilege escalation tactic consists of techniques that are used by attackers to get higher privileges in the environment than those they currently have. In containerized environments, this can include getting access to the node from a container, gaining higher privileges in the cluster, and even getting access to the cloud resources.

Techniques	Mitigations
Privileged container	**MS-M9013**: Restrict over permissive containers **MS-M9017**: Ensure that pods meet defined Pod Security Standards **MS-M9005.003**: Gate images deployed to Kubernetes cluster
Cluster-admin binding	**MS-M9003**: Adhere to least-privilege principle
Writable hostPath mount	**MS-M9013**: Restrict over permissive containers **MS-M9016**: Restrict File and Directory Permissions **MS-M9011**: Restrict Container runtime using LSM **MS-M9017**: Ensure that pods meet defined Pod Security Standards
Access cloud resources	**MS-M9003**: Adhere to least-privilege principle **MS-M9018**: Restrict the access of pods to IMDS **MS-M9019**: Allocate specific identities to pods **MS-M9013**: Restrict over permissive containers

Table 3.4: Privilege Escalation Tactic

Defense Evasion

A set of techniques employed by attackers to elude detection and conceal their actions.

Techniques	Mitigations
Clear container logs	**MS-M9020**: Collect Logs to Remote Data Storage **MS-M9016**: Restrict File and Directory Permissions
Delete Kubernetes events	**MS-M9020**: Collect Logs to Remote Data Storage **MS-M9003**: Adhere to least-privilege principle
Pod or container name similarity	**MS-M9005.003**: Gate images deployed to Kubernetes cluster
Connect from proxy server	**MS-M9002**: Restrict access to the API server using IP firewall **MS-M9014**: Network Segmentation **MS-M9021**: Restrict the usage of unauthenticated APIs in the cluster **MS-M9009**: Require Strong Authentication to Services

Table 3.5: Defense Evasion Tactic

Credential Access

The credential access tactic consists of techniques that are used by attackers to steal credentials.

In containerized environments, this includes credentials of the running application, identities, secrets stored in the cluster, or cloud credentials.

Techniques	Mitigations
List Kubernetes secrets	**MS-M9003**: Adhere to least-privilege principle **MS-M9022**: Use Managed Secret Store **MS-M9023**: Remove unused secrets objects from the cluster **MS-M9024**: Restrict access to etcd
Mount service principal	**MS-M9013**: Restrict over permissive containers **MS-M9003**: Adhere to least-privilege principle
Container service account	**MS-M9025**: Disable Service Account Auto Mount **MS-M9003**: Adhere to least-privilege principle
Credentials in configuration files	**MS-M9026**: Avoid using plain text credentials **MS-M9022**: Use Managed Secret Store
Malicious admission controller	**MS-M9003**: Adhere to least-privilege principle

Table 3.6: Credential Access Tactic

Discovery

The discovery tactic consists of techniques that are used by attackers to explore the environment to which they gained access. This exploration helps the attackers to perform lateral movement and gain access to additional resources.

Techniques	Mitigations
Access Kubernetes API server	**MS-M9003**: Adhere to least-privilege principle **MS-M9002**: Restrict access to the API server using IP firewall
Access Kubelet API	**MS-M9009**: Require Strong Authentication to Services **MS-M9014**: Network Segmentation **MS-M9003**: Adhere to least-privilege principle **MS-M9027**: Use NodeRestriction Admission Controller
Network mapping	**MS-M9014**: Network Segmentation
Exposed sensitive interfaces	**MS-M9008**: Limit Access to Services Over Network **MS-M9009**: Require Strong Authentication to Services **MS-M9014**: Network Segmentation
Instance Metadata API	**MS-M9018**: Restricting cloud metadata API access
Access host filesystem	**MS-M9013**: Restrict over permissive containers **MS-M9016**: Restrict File and Directory Permissions **MS-M9011**: Restrict Container runtime using LSM **MS-M9017**: Ensure that pods meet defined Pod Security Standards

Table 3.7: Discovery Tactic

Lateral Movement

The lateral movement tactic consists of techniques that are used by attackers to move through the victim's environment. In containerized environments, this includes gaining access to various resources in the cluster from a given access to one container, gaining access to the underlying node from a container, or gaining access to the cloud environment.

Techniques	Mitigations
Access cloud resources	**MS-M9003**: Adhere to least-privilege principle **MS-M9018**: Restrict the access of pods to IMDS **MS-M9019**: Allocate specific identities to pods **MS-M9013**: Restrict over permissive containers
Container service account	**MS-M9025**: Disable Service Account Auto Mount **MS-M9003**: Adhere to least-privilege principle
Cluster internal networking	**MS-M9014**: Network Segmentation **MS-M9005**: Image Assurance Policy
Credentials in configuration files	**MS-M9026**: Avoid using plain text credentials **MS-M9022**: Use Managed Secret Store
Writable hostPath mount	**MS-M9013**: Restrict over permissive containers **MS-M9016**: Restrict File and Directory Permissions **MS-M9011**: Restrict Container runtime using LSM **MS-M9017**: Ensure that pods meet defined Pod Security Standards
CoreDNS poisoning	**MS-M9003**: Adhere to least-privilege principle
Access host filesystem	**MS-M9013**: Restrict over permissive containers **MS-M9028**: Use CNIs that are not prone to ARP poisoning

Table 3.8: Lateral Movement Tactic

Collection

Collection in Kubernetes consists of techniques that are used by attackers to collect data from the cluster or through using the cluster.

Techniques	Mitigations
Images from a private registry	**MS-M9018**: Restricting cloud metadata API access **MS-M9003**: Adhere to least-privilege principle
Collecting data from pod	**MS-M9003**: Adhere to least-privilege principle **MS-M9010**: Restrict Exec Commands on Pods

Table 3.9: Collection Tactic

Impact

The Impact tactic consists of techniques that are used by attackers to destroy, abuse, or disrupt the normal behavior of the environment.

Techniques	Mitigations
Data destruction	**MS-M9030**: Use Cloud Storage Provider **MS-M9031**: Implement Data Backup Strategy
Resource hijacking	**MS-M9011**: Restrict Container Runtime using LSM **MS-M9012**: Remove Tools from Container Images
Denial of service	**MS-M9011**: Restrict Container Runtime using LSM **MS-M9002**: Restrict access to the API server using IP firewall **MS-M9029**: Set requests and limits for containers

Table 3.10: Impact Tactic

3.1.2 List of mitigations

As each mitigation can be applied to one or more tactics, the chosen approach will be to review existing solutions for each mitigation regardless of the tactic associated.

Multi-factor authentication

ID	MITRE mitigation	Description
MS-M9001	M1032	Using multi-factor authentication for accounts can prevent unauthorized access in case an adversary achieves access to the account credentials. This can reduce the risk in case an adversary achieved valid credentials to an account that has permissions to the Kubernetes cluster.

Multi-factor authentication is not possible only with *plain* Kubernetes, but it can be accomplished by the use of a third-party element:

- Dex is "an identity service that uses OpenID Connect to drive authentication for other apps"[23]. It defers authentication via *connectors* to LDAP servers, SAML providers, or established identity providers like GitHub, Google, and Active Directory.
- Pinniped is VMware solution for providing identity services to Kubernetes([24]) as part of their VMware Tanzu project[25].
- When deploying in a cloud provider environment you can make use of their existing authentication service, for example:
  - Elastic Kubernetes Service (EKS) can make use of Amazon Web Services (AWS) IAM[26]
  - Google Kubernetes Engine (GKE) can make use of Google Cloud Provider (GCP) Google OAuth[27]
  - Azure Kubernetes Service (AKS) can make use of Azure Active Directory [28]

Restrict access to the API server using IP firewall

ID	MITRE mitigation	Description
MS-M9002	M1035	Restricting access to the API server can prevent unwanted access to the clusters management, even if the adversary achieved valid credentials to the cluster. In managed clusters, cloud providers often support native built-in firewall which can restrict the IP addresses that are allowed to access the API server.

Restricting access to the API server using IP firewall serves multiple scenarios:

- The attacker can interact with the API server through a vulnerability

- The attacker has valid credentials
- The API server allows anonymous access

If the company manages its own networks, it can use its own firewalls. In our case, as we work in a Cloud environment, the options are to firewall the API via EC2 Security Groups or setting the cluster endpoint to private in the case of AWS EKS.

Adhere to least-privilege principle

ID	MITRE mitigation	Description
MS-M9003	M1018	Configure the Kubernetes role-based access controls (RBAC) for each user and service accounts to have only necessary permissions.

In order to reduce the impact of an attack, users should be given only the necessary permissions. CIS Controls[9] section 5.1 *RBAC and Service Accounts*. The controls are:

- **5.1.1**: Ensure that the cluster-admin role is only used where required
- **5.1.2**: Minimize access to secrets
- **5.1.3**: Minimize wildcard use in Roles and ClusterRoles
- **5.1.4**: Minimize access to create pods
- **5.1.5**: Ensure that default service accounts are not actively used
- **5.1.6**: Ensure that Service Account Tokens are only mounted where necessary

The qualitative nature of some of these controls makes them difficult to enforce beyond compliance reports. However, in situations where it is known that certain scenarios will never happen, a third-party software like OPA Gatekeeper can prevent the creation or modification of suspicious Role-Based Access Control (RBAC) objects through the use of an appropriate Rego rule.

Secure CI/CD environment

ID	MITRE mitigation	Description
MS-M9004	-	Security code repositories and CI/CD environment by placing gates to restrict unauthorized access and modification of content. This can include enforcing RBAC permissions to access and make changes to code, artifacts and build pipelines, ensure governed process for pull-request approval, apply branch policies and others.

Image assurance policy

ID	MITRE mitigation	Description
MS-M9005	M1016, M1045	Apply image assurance policy to evaluate container images against vulnerabilities, malware, exposed secrets or other policies.

There are several products on the market that scans your container images (manually or automatically) but an open-source project like Trivy[29] can also be used.

Gate generated images in CI/CD pipeline

ID	MITRE mitigation	Description
MS-M9005.001	M1016, M1045	Placing gates in the CI/CD pipeline that can cancel or fail pipeline execution to block container images not meeting content trust requirements.

Gate images pushed to registries

ID	MITRE mitigation	Description
MS-M9005.002	M1016, M1045	Placing gates in the container registry to prevent pushing or quarantine images that does not meet the content trust requirement.

Some container registries can support gates that will prevent pushing images, while others might quarantine images after they were already push to the registry. Ensuring that gates exists at the registry level can help preventing bypass of gates at the CI/CD pipelines level.

Gate images deployed to Kubernetes cluster

ID	MITRE mitigation	Description
MS-M9005.003	M1016, M1045	Gate deployment of images to Kubernetes cluster to prevent deploying images that does not meet the content trust requirements.

This can include limiting images to be deployed only from trusted registries, to have digital signature or pass vulnerability scanning and other checks. This can prevent potential adversaries from using their own malicious images in the cluster. Also, this ensures that only images that passed the security compliance policies of the organization are deployed in the cluster. Kubernetes admission controller mechanism is one of the commonly used tools for implementing such policy, for example with OPA Gatekeeper[30].

Enable Just In Time access to API server

ID	MITRE mitigation	Description
MS-M9006	-	Employing Just In Time (JIT) elevated access to Kubernetes API server helps reduce the attack surface to the API server by compromised accounts by allowing access only at specific times, and through a governed escalation process.

Enabling JIT access in Kubernetes is often done together with OpenID authentication which includes processes and tools to manage JIT access. One example of such OpenID authentication is Azure Active Directory authentication to Kubernetes clusters. The JIT approval is performed in the cloud controlplane level. Therefore, even if attackers have access to an account credentials, their access to the cluster is limited.

Network intrusion prevention

ID	MITRE mitigation	Description
MS-M9007	M1031	Use intrusion detection signatures and web application firewall to block traffic at network boundaries to pods and services in a Kubernetes cluster.

Adapting the network intrusion prevention solution to Kubernetes environment might be needed to route network traffic destined to services through it. In some cases, this will be done by deploying a containerized version of a network intrusion prevention solution to the Kubernetes cluster and be part of the cluster network, and in some cases, routing ingress traffic to Kubernetes services through an external appliance, requiring that all ingress traffic will only come from such an appliance.

Two products that cover this mitigation, both based on the Extended Berkeley Packet Filter technology (eBPF[31]) observability, are:

- Falco[32] (detection only, automated response through *Sysdig Secure*)
- Tetragon[33] (detection and reaction to events such as process execution events, system call activity and I/O activity)

Limit access to services over network

ID	MITRE mitigation	Description
MS-M9008	M1035	Avoid exposing sensitive interfaces insecurely to the Internet or limit access to it. Sensitive interfaces includes management tools and applications that allow creation of new containers in the cluster.

Some of those services does not use authentication by default and are not intended to be exposed. Examples of services that were exploited: Weave Scope, Apache NiFi and more.

If services need to be exposed to the internet and are exposed using LoadBalancer service, use IP restriction (loadBalancerSourceRanges) when possible. This reduces the attack surface of the application and can prevent attackers from being able to reach the sensitive interfaces.

Require strong authentication to services

ID	MITRE mitigation	Description
MS-M9009	-	Use strong authentication when exposing sensitive interfaces to the Internet.

For example, attacks were observed against exposed Kubeflow and Argo workloads that were not configured to use OpenID Connect or other authentication methods.

Use strong authentication methods to the Kubernetes API that will prevent attackers from gaining access to the cluster even if valid credentials such as kubeconfig were achieved. For example, in AKS use AAD authentication instead of basic authentication. By using AAD authentication, a short-lived credential of the cluster is retrieved after authenticating to AAD.

Avoid using the read-only endpoint of Kubelet in port 10255, which doesn't require authentication. In newer version of managed clusters, this port is disabled.

Restrict exec commands on pods

ID	MITRE mitigation	Description
MS-M9010	-	Restrict running Kubenetes exec command on sensitive/production containers using admission controller. This can prevent attackers from running malicious code on containers in cases when he pods/exec permission was obtained.

This can be controlled via RBAC and eBPF products like Tetragon.

Restrict container runtime using LSM

ID	MITRE mitigation	Description
MS-M9011	M1038, M1040	Restrict the running environment of the containers using Linux security modules, such as AppArmor, SELinux, Seccomp and others. Linux security modules can restrict access to files, running processes, certain system calls and others. Also, dropping unnecessary Linux capabilities from the container runtime environment helps reduce the attack surface of such container.

An implementation for this mitigation would imply hardening the Kubernetes nodes, using a solution like Tetragon (LSM) and/or OPA Gatekeeper (pod capabilities enforcing).

Remove tools from container images

ID	MITRE mitigation	Description
MS-M9012	M1042	Attackers often use built-in executables to run their malicious code. Removing unused executables from the image filesystem can prevent such activity.

Examples of executables that are commonly used in malicious activity include: sh, bash, curl, wget, chmod and more.

Restrict over permissive containers

ID	MITRE mitigation	Description
MS-M9013	M1038	Use admission controller to prevent deploying containers with over-permissive capabilities or configuration in the cluster. This can include restricting privileged containers, containers with sensitive volumes, containers with excessive capabilities, and other signs of over permissive containers.

Outside of promoting best practices within the organization, Kubernetes admission controllers (for example OPA Gatekeeper) can be used to make sure pods don't run with excessive capabilities.

Network segmentation

ID	MITRE mitigation	Description
MS-M9014	M1030	Restrict inbound and outbound network traffic of the pods in the cluster. This includes inner-cluster communication as well as ingress/egress traffic to/from the cluster. Network Policies are a native K8s solution for networking restrictions in the cluster.

Service meshes and CNI plugins can help if there is a need for fine-grained controls

Avoid running management interface on containers

ID	MITRE mitigation	Description
MS-M9015	M1042	Avoid running SSH daemon, as well as other management interfaces, if they aren't necessary for the application's functionality.

Restrict file and directory permissions

ID	MITRE mitigation	Description
MS-M9016	M1022	When using hostPath volumes, set it to "read-only" mode if possible. This prevents the container from writing to files in the underlying node and will harden an escape from the container to the node.

Outside of promoting best practices within the organization, Kubernetes admission controllers (for example OPA Gatekeeper) can be used to make sure pods don't run with excessive capabilities.

Ensure that pods meet defined Pod Security Standards

ID	MITRE mitigation	Description
MS-M9017	-	The Pod Security Standards define three different policies to broadly cover the security spectrum. These policies are cumulative and range from highly-permissive to highly-restrictive.

Decoupling policy definition from policy instantiation allows for a common understanding and consistent language of policies across clusters, independent of the underlying enforcement mechanism. At the same time, Kubernetes offers a built-in Pod Security admission controller to enforce the Pod Security Standards. Pod security restrictions are applied at the namespace level when pods are created.

Restricting cloud metadata API access

ID	MITRE mitigation	Description
MS-M9018	M1035	Many cluster-to-cloud authentication methods involve access to the node's metadata server. Restrict access to the metadata server if it's not necessary.

This can be done at the pod level by using networking restriction tools such as network policies. Alternatively, cloud providers allow this functionality in the node/cluster level.

Allocate specific identities to pods

ID	MITRE mitigation	Description
MS-M9019	-	When needed, allocate dedicated cloud identity per pod with minimal permissions, instead of inheriting the node's cloud identity. This prevents other pods from accessing cloud identities that are not necessary for their operation.

The features that implement this separation are: Azure AD Pod Identity (AKS), Azure AD Workload identity (AKS), IRSA (EKS) and GCP Workload Identity (GCP).

Collect logs to remote data storage

ID	MITRE mitigation	Description
MS-M9020	M1029	Collect the Kubernetes and application logs of pods to external data storage to avoid tampering or deletion.

This can be achieved by various open-source tools such as Fluentd. Also, built-in cloud solutions are available for managed clusters, such as Container Insights and Log Analytics in AKS and Cloud Logging in GKE

Restrict the usage of unauthenticated APIs in the cluster

ID	MITRE mitigation	Description
MS-M9021	-	Some unmanaged clusters are misconfigured such as anonymous access is accepted by the Kubernetes API server. Make sure that the Kubernetes API is configured properly, and authentication and authorization mechanisms are set.

This is managed by CIS control 4.2.1. Also mitigated by using a cloud provider managed cluster.

Use managed secret store

ID	MITRE mitigation	Description
MS-M9022	M1029	Use cloud secret store, such as Azure Key Vault, to securely store secrets that are used by the workloads in the cluster.

This allows cloud-level management of the secret which includes permission management, expiration management, secret rotation, auditing, etc. The integration of cloud secret stores with Kubernetes is done by using Secrets Store CSI Driver, which is implemented by all major cloud providers.

Remove unused secrets from the cluster

ID	MITRE mitigation	Description
MS-M9023	-	Remove unused secrets objects from the cluster.

Restrict access to etcd

ID	MITRE mitigation	Description
MS-M9024	M1035	Access to etcd should be limited to the Kubernetes control plane only.

Depending on your configuration, you should attempt to use etcd over TLS. This mitigation is relevant only to non-managed Kubernetes environment, as access to etcd in cloud managed clusters is already restricted.

Disable service account auto mount

ID	MITRE mitigation	Description
MS-M9025	-	By default, a service account is mounted to every pod. If the application doesn't require access to the Kubernetes API, disable the service account auto-mount by specifying automountServiceAccount-Token: false in the pod configuration.

Outside of promoting best practices within the organization, Kubernetes admission controllers (for example OPA Gatekeeper) can be used to make sure pods don't auto-mount a service account.

Avoid using plain text credentials

ID	MITRE mitigation	Description
MS-M9026	-	Avoid using plain text credentials in configuration files. Use Kubernetes secrets or cloud secret store instead. This prevents unwanted access to plaintext credentials in source code, configuration files and Kubernetes objects.

There are several products on the market that scan your configuration files (manually or automatically) but you can also use an open-source project like Trivy[29].

Use NodeRestriction admission controller

ID	MITRE mitigation	Description
MS-M9027	-	NodeRestriction admission controller limits the permissions of kubelet and allows it to modify only its own Node object and only the pods that are running on its own node.

This may limit attackers who have access to the Kubelet API from gaining full control over the cluster.

Use CNIs that are not prone to ARP poisoning

ID	MITRE mitigation	Description
MS-M9028	-	Kubernetes default CNI (Kubenet) is prone to ARP poisoning. This allows pods to impersonate other pods in the cluster. Use alternative CNIs that are not prone to ARP poisoning in the cluster.

Use alternative CNIs that are not prone to ARP poisoning in the cluster.

Set requests and limits for containers

ID	MITRE mitigation	Description
MS-M9029	-	Set requests and limits for each container to avoid resource contention and DoS attacks.

Outside of promoting best practices within the organization, Kubernetes admission controllers (for example OPA Gatekeeper) can be used to make sure requests and limits are defined.

Use cloud storage provider

ID	MITRE mitigation	Description
MS-M9030	-	Use cloud storage services, such as Azure Files, for storing the application's data.

Kubernetes integrates with all main cloud provider storage services as storage providers for pod volumes. This allows leveraging cloud storage capabilities such as backup and snapshots.

Implement data backup strategy

ID	MITRE mitigation	Description
MS-M9031	-	Take and store data backups from pod mounted volumes for critical workloads.

Ensure backup and storage systems are hardened and kept separate from the Kubernetes environment to prevent compromise.

Avoid using web-hosted manifest for Kubelet

ID	MITRE mitigation	Description
MS-M9032	-	Kubelet can deploy static pods by using manifests that are stored in web accessible locations.

If web-hosted manifest are not required, make sure that Kubelet does not run with –manifest-url argument.

Chapter 4

Implementation

The focus of this thesis is on securing a deployed Kubernetes cluster in a public cloud environment. As such, the study case will involve the security status of a v1.24 Kubernetes cluster in the AWS cloud provider. Setting a specific Kubernetes version enables us to align with existing benchmarks. The selection of AWS as the cloud provider is driven by the same rationale and serves only to provide specific implementations.

4.1 Case Study

The company we are going to establish as our starting point is a small company of about thirty employees. The product they develop is a social marketplace in which train-modeling enthusiasts buy and sell models and tools of the craft. The software is only available in web form, with no mobile applications. Lately, they have been having good reception and they are looking to expand, wich will imply growing the number of developers working in the company, and for sure atracting more attention as the popularity grows. Management, which until now did not pay too much attention to their security posture, is beginning to worry about security and regulations and have asked the team about possible avenues to improve cybersecurity.

The Kubernetes cluster is deployed in AWS on top of EC2 virtual machines, and its API is accessible to the internet. Developers interact directly from their computers with their tools of choice (*kubectl*, *k9s*, *OpenLens*), and they are given the option of working in the office or remotely. Developers are still handed the admin credentials to the cluster, a practice put in place when the company was small and everyone knew each other well. The end users only access a webpage which is the *frontend* for the whole system, comprised of tens of microservices. They host their code in Github and build and push their software via a Jenkins instance deployed outside the Kubernetes cluster.

4.1.1 Establishing Priorities

Our attack vectors are defined by our attack surface, which means that our priority will always be to reduce the surface rather than react to potential attacks. Once we can't reduce the attack surface anymore, we will then focus on mitigating the remaining threat vectors.

To this end, we can follow the next flowchart:

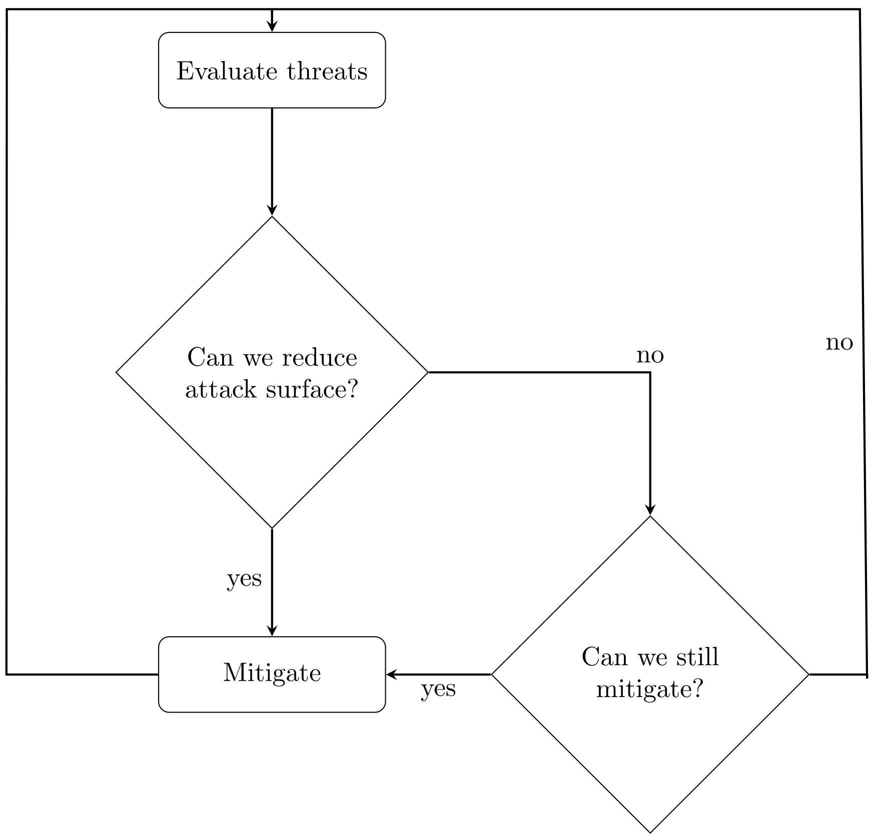

Figure 4.1: Priority evaluation

When is it not possible to mitigate?

Every action has a cost, whether it is a direct expense such as the price of a tool, or an indirect one such as the time spent by an employee, so ultimately it is up to the company's discretion to decide which mitigations to implement. In the context of this work, we will consider the proposed implementations as recommendations, since it is beyond the scope to economically compare their cost against a budget.

4.2 Domains of application

To be able to prioritize the work to be done, we have to know the level of expertise they require and the effort they need. To this end the goal of this section is to go over implementations to give coverage to the Microsoft Threat Matrix for Kubernetes. Once we have the information about the tasks to be done, we will be able to prioritize.

We identify three actionable domains at which to implement mitigations:

- **Kubernetes configurations:** This cover all configurations available for a *vanilla* Kubernetes cluster.
- **Cloud Provider configuration:** Configurations made at the cloud provider level. This includes installing software in the cluster that integrates with the provider services
- **Third-party software:** External software which when installed in the cluster, provides functionality

The following table shows the domains in which they can be applied:

Mitigation	Domain
Multi-factor authentication	Cloud Provider
Restrict access to the API server using IP firewall	Cloud Provider
Adhere to least-privilege principle	Kubernetes configuration
Image assurance policy	Third-party software
Gate images deployed to Kubernetes cluster	Third-party software
Enable Just In Time access to API server	Cloud Provider, Third-party software
Network intrusion prevention	Third-party software
Limit access to services over network	Cloud Provider
Restrict exec commands on pods	Kubernetes configuration, Third-party software
Restrict container runtime using LSM	Third-party software
Restrict over permissive containers	Third-party software
Network segmentation	Cloud Provider
Restrict file and directory permissions	Third-party software
Ensure that pods meet defined Pod Security Standard	Third-party software
Restricting cloud metadata API access	Cloud Provider
Allocate specific identities to pods	Cloud Provider
Collect logs to remote data storage	Cloud Provider
Restrict the usage of unauthenticated APIs in the cluster	Cloud Provider

Mitigation	Domain
Use managed secret store	Cloud Provider
Remove unused secrets from the cluster	Third-party software
Restrict access to etcd	Cloud Provider, Kubernetes configuration
Disable service account auto mount	Third-party software
Use NodeRestriction admission controller	Kubernetes configuration
Use CNIs that are not prone to ARP poisoning	Third-party software
Set requests and limits for containers	Third-Party software
Use cloud storage provider	Cloud provider
Implement data backup strategy	Cloud provider
Avoid using web-hosted manifest for Kubelet	Kubernetes configuration

It is worth noting that certain measures to secure a Kubernetes cluster in a cloud environment are beyond the scope of this thesis fall out of scope due to being part of the development process or CI/CD. These mitigations are:

- Secure CI/CD environment
- Gate generated images in CI/CD pipeline
- Gate images pushed to registries
- Remove tools from container images
- Avoid running management interface on containers
- Avoid using plain text credentials
- Require strong authentication to services

The following sections are the mitigations implementation for each domain. Please note that they are not the only way to be implemented. Instead, each mitigation has been chosen with the following goals in mind:

- Open source availability
- The value they bring in a effort/benefit scale
- Extendibility of the implementations

4.2.1 Kubernetes configuration

Adhere to least-privilege principle

"The RBAC API declares four kinds of Kubernetes object: Role, ClusterRole, RoleBinding and ClusterRoleBinding"[34]. By creating *Roles* or *ClusterRoles*with only the necessary permissions and binding them to accounts, least-privileged access can be provided.

Restrict exec commands on pods

The Kubernetes resource that controls if *exec commands* on pods are allowed is "*pods/exec*". Following the least-privilege principle, make sure that permissions to this resource are not granted if is not necessary.

Use NodeRestriction admission controller

"The NodeRestriction admission plugin prevents kubelets from deleting their Node API object, and enforces kubelet modification of labels under the *kubernetes.io/* or *k8s.io/*"[35].

Avoid using web-hosted manifest for Kubelet

EKS provides a way to configure the Kubernetes Kubelet[36] via the *config file* Custom Resource Definition (CRD)[37]. When deploying this CRD, make sure that in the *nodeGroups.kubeletExtraConfig* key there is no manifestUrl key set up. An example of a configuration could be as follows:

```yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: dev-cluster-1
  region: eu-north-1

nodeGroups:
  - name: ng-1
    instanceType: m5a.xlarge
    desiredCapacity: 1
    kubeletExtraConfig:
        kubeReserved:
            cpu: "300m"
            memory: "300Mi"
            ephemeral-storage: "1Gi"
        kubeReservedCgroup: "/kube-reserved"
        systemReserved:
            cpu: "300m"
            memory: "300Mi"
            ephemeral-storage: "1Gi"
        evictionHard:
            memory.available: "200Mi"
            nodefs.available: "10%"
```

```
featureGates:
  RotateKubeletServerCertificate: true # has to be enabled,
      otherwise it will be disabled
```

Listing 4.1: EKS Kubelet configuration

4.2.2 Cloud Provider configuration

Multi-factor authentication

In order to prevent unauthorized access in case an adversary achieves access to account credentials, one option is to use Multi-factor authentication. As in our case we are running inside Amazon Web Services, we can make use of *Multi-Factor Authentication (MFA) for IAM*[38]. The users that connect to the cluster should have the option enabled and use the supported mechanisms:

- FIDO security key[39]
- Virtual MFA devices[40]
- Hardware TOTP tokens[41]

Note that the mechanisms are not exclusive and it is recommended to enable multiple MFA devices. Once the users have the necessary authentication settings enabled, we can then apply the IAM roles that will let them connect to the Kubernetes cluster.

Restrict access to the API server using IP firewall

Amazon EKS cluster endpoint access control lets you enable private access and limit, or completely disable, public access from the internet[42]. Effectively, this lets you choose from the following possibilities:

- Public access enabled, Private access disabled
  - This is the default behavior for new Amazon EKS clusters
  - Kubernetes API requests that originate from within your cluster's VPC (such as node to control plane communication) leave the VPC but not Amazon's network.
  - Your cluster API server is accessible from the internet. You can, optionally, limit the CIDR blocks that can access the public endpoint. If you limit access to specific CIDR blocks, then it is recommended that you also enable the private endpoint, or ensure that the CIDR blocks that you specify include the addresses that nodes and Fargate Pods (if you use them) access the public endpoint from.
- Public access enabled, Private access enabled

  - Kubernetes API requests within your cluster's VPC (such as node to control plane communication) use the private VPC endpoint.
  - Your cluster API server is accessible from the internet. You can, optionally, limit the CIDR blocks that can access the public endpoint.

- Public access disabled, Private access enabled
  - All traffic to your cluster API server must come from within your cluster's VPC or a connected network.
  - There is no public access to your API server from the internet. Any kubectl commands must come from within the VPC or a connected network.
  - The cluster's API server endpoint is resolved by public DNS servers to a private IP address from the VPC.

While it is true that with public access enabled we can limit the CIDR blocks that can access the public endpoint, ideally the cluster endpoint should be private, and the users that need access should be accessing the cluster via a bastion host inside the VPC or via a network that is connected with an AWS transit gateway or other connectivity option. With this in mind, the endpoint can be configured with the following command:

```
aws eks update-cluster-config \
   --region region-code \
   --name my-cluster \
   --resources-vpc-config endpointPublicAccess=false,
       endpointPrivateAccess=true
```

Listing 4.2: Amazon EKS cluster endpoint access control

Enable Just In Time access to API server

Adhering to the least-privilege principle, users should not have more permissions that they wouldn't need in their day-to-day work. There are cases though where a user needs elevated permissions, for example during an incident where no administrators are present or available. In this case, a break-glass mechanism can be set up that grants elevated permissions temporarily. Amazon Web Services provides a sample architecture for this case in a GitHub repository called *aws-iam-temporary-elevated-access-broker*[43] in their *aws-samples* organization. The solution consists of:

- A web application ("app UI") that runs in the browser, known as a Single Page Application (SPA)
- A CloudFront distribution to serve static content
- Server-side APIs hosted by Amazon API Gateway and AWS Lambda

- A DynamoDB table to track the status of temporary elevated access requests

The architecture is as follows:

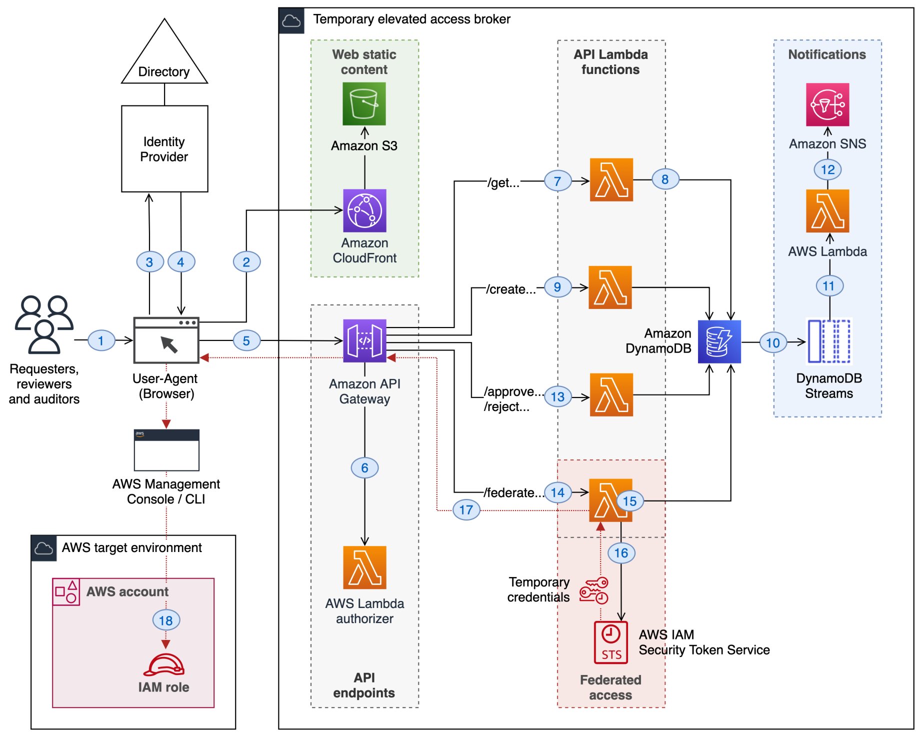

Figure 4.2: A minimal reference implementation for temporary elevated access

Limit access to services over network

In an unexposed Kubernetes cluster running in AWS you can expose your services via an AWS Network Load Balancer[44]. For this you have to create a Kubernetes Service of type LoadBalancer and specify the appropriate annotations to configure it. An example for an internet facing service is as follows:

```
apiVersion: v1
kind: Service
metadata:
  name: nlb-sample-service
  namespace: nlb-sample-app
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
```

```yaml
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
    app: nginx
```

Listing 4.3: Internet facing service via Network Load Balancer

The annotations you can specify cover multiple areas, among them:

- Traffic Routing
- Traffic Listening
- Resource attributes
- Access control

The Kubernetes SIGS group maintains extensive documentation about these annotations[45].

Restricting cloud metadata API access

The EC2 Instance Metadata Service (IMDS) is accessible to all EC2 instances by default. This service provides useful introspection facilities, such as determining a node's availability zone, instance ID, and so forth. In addition, IMDS provides access to IAM credentials that allow applications to assume the instance's IAM role.

One way to block pod IMDS access is to require IMDS version 2 (IMDSv2) to be used, and to set the maximum hop count to 1. Configuring IMDS this way will cause requests to IMDS from pods to be rejected, provided those pods do not use host networking.[46]

Another way to block pod IMDS is through the use of network policies to ensure pods are unable to reach the Instance Metadata Service. To do this, configure your network policy to block egress traffic to `169.254.0.0/16`. As per the Kubernetes Network Policy model, Cilium policies follow the whitelist model[47]. When a policy is enabled for a pod, all ingress and egress traffic are denied by default unless the policy specification allows specific traffic. As a result, inter-namespace communication will be denied by default and we need policy specifications to whitelist traffic within a namespace and legitimate traffic in and out of a namespace. To avoid accessing IMDS, leave out `169.254.0.0/16` in the egress whitelist:

```yaml
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "cidr-rule"
spec:
  endpointSelector:
```

```
    matchLabels:
      app: myService
  egress:
  - toCIDR:
    - 20.1.1.1/32
  - toCIDRSet:
    - cidr: 10.0.0.0/8
      except:
      - 10.96.0.0/12
```

Listing 4.4: Cilium Network Policy

Allocate specific identities to pods

With IAM Roles for Service Accounts (IRSA) allows you to assign an IAM Role to a Kubernetes service account[48]. It works by leveraging a Kubernetes feature known as Service Account Token Volume Projection. When Pods are configured with a Service Account that references an IAM Role, the Kubernetes API server will call the public OIDC discovery endpoint for the cluster on startup. The endpoint cryptographically signs the OIDC token issued by Kubernetes and the resulting token mounted as a volume. This signed token allows the Pod to call the AWS APIs associated IAM role. When an AWS API is invoked, the AWS SDKs calls `sts:AssumeRoleWithWebIdentity`. After validating the token's signature, IAM exchanges the Kubernetes issued token for a temporary AWS role credential.

To associate an existing IAM role to a Kubernetes Service Account, annotate the object with the `eks.amazonaws.com/role-arn` key[49]:

```
kubectl annotate serviceaccount -n $namespace $service_account eks.
    amazonaws.com/role-arn=arn:aws:iam::$account_id:role/my-role
```

Listing 4.5: Assign IAM role to service account

Restrict access to etcd

This mitigation is relevant only to non-managed Kubernetes environment, as access to etcd in cloud managed clusters is already restricted.

Restrict the usage of unauthenticated APIs in the cluster

By using AWS EKS this mitigation is already implemented as EKS requires all API requests to be authenticated[42].

Use cloud storage provider and Implement data backup strategy

"The Amazon Elastic Block Store (Amazon EBS) Container Storage Interface (CSI) driver allows Amazon Elastic Kubernetes Service (Amazon EKS) clusters to manage the lifecycle of Amazon EBS volumes for persistent volumes"[52]. The EBS CSI driver requires an IAM role with a special policy to function. To create the role:

```
eksctl create iamserviceaccount \
  --name ebs-csi-controller-sa \
  --namespace kube-system \
  --cluster my-cluster \
  --attach-policy-arn arn:aws:iam::aws:policy/service-role/
      AmazonEBSCSIDriverPolicy \
  --approve \
  --role-only \
  --role-name AmazonEKS_EBS_CSI_DriverRole
```

Listing 4.6: IAM role creation

After that create the policy and attach it to the role:

```
aws iam create-policy \
  --policy-name KMS_Key_For_Encryption_On_EBS_Policy \
  --policy-document file://kms-key-for-encryption-on-ebs.json
```

Listing 4.7: IAM policy creation

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::111122223333:policy/
      KMS_Key_For_Encryption_On_EBS_Policy \
  --role-name AmazonEKS_EBS_CSI_DriverRole
```

Listing 4.8: IAM policy attachment

Then the CSI driver can be installed via eksctl by executing:

```
eksctl create addon \
  --name aws-ebs-csi-driver \
  --cluster mycluster \
  --service-account-role-arn arn:aws:iam::[ID]:role/
      AmazonEKS_EBS_CSI_DriverRole \
  --force
```

Listing 4.9: EBS CSI driver addon installation

While it is ideal that every application manage their backup procedures, once we are using the AWS EBS CSI, snapshots of the mounted volumes for critical workloads can be scheduled. If the volume wasn't encrypted, the snapshot process can encrypt it on the fly, increasing its protection[53].

Collect logs to remote data storage

Control plane logging can be enabled by executing[54]:

```
aws eks update-cluster-config \
  --region region-code \
  --name my-cluster \
  --logging '{"clusterLogging":[{"types":["api","audit","authenticator
      ","controllerManager","scheduler"],"enabled":true}]}'
```

Listing 4.10: Amazon EKS control plane logging

AWS also provides a solution that sends workload logs to CloudWatch[55] via Fluent Bit[56].

Use managed secret store

Despite their name, secrets in a vanilla Kubernetes cluster are just base64-encoded strings. To improve the security of our secrets, which can hold sensitive data, we can integrate the AWS Secret Manager to show secrets and parameters from Parameter Store as files mounted in the Amazon EKS Pods. For this, we can use the AWS Secrets and Configuration Provider (ASCP) for the Kubernetes Secrets Store Container Storage Interface (CSI) Driver[50].

To describe which files to create in the Amazon EKS pod and which secrets to put in them, you create a SecretProviderClass YAML file. The SecretProviderClass must be in the same namespace as the Amazon EKS pod it references. If you use Secrets Manager automatic rotation for your secrets, you can also use the Secrets Store CSI Driver rotation reconciler feature to ensure you are retrieving the latest secret from Secrets Manager. Once the ASCP is installed and configured[51] you can mount key/values from a secret like so:

```yaml
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: aws-secrets
spec:
  provider: aws
  parameters:
    objects: |
      - objectName: "arn:aws:secretsmanager:us-east-2:111122223333:secret
          :MySecret-a1b2c3"
        jmesPath:
            - path: username
              objectAlias: dbusername
            - path: password
              objectAlias: dbpassword
```

Listing 4.11: Mount a secret from Secret Manager store

4.2.3 Third-party software

Trivy

Image assurance policy

Trivy[57] is a well-established security scanner that can be used to scan:

- Container Images
- Filesystems

- Git Repositories (remote)
- Virtual Machine Images
- Kubernetes
- AWS

It is an open-source project maintained by AquaSecurity that is usually executed proactively be it manually or as part of a CI/CD pipeline. Its scanners modules can find:

- OS packages and software dependencies in use (SBOM)
- Known vulnerabilities (CVEs)
- IaC issues and misconfigurations
- Sensitive information and secrets
- Software licenses

In May 2022, Aqua announced the Trivy Kubernetes operator[59]. Following the Kubernetes *controller*[58] pattern, it automatically supdates security reports in response to workload and other changes on a Kubernetes cluster, generating the following reports:

- Vulnerability Scans: Automated vulnerability scanning for Kubernetes workloads
- ConfigAudit Scans: Automated configuration audits for Kubernetes resources with predefined rules or custom Open Policy Agent (OPA) policies
- Exposed Secret Scans: Automated secret scans which find and detail the location of exposed Secrets within your cluster
- RBAC scans: Role Based Access Control scans provide detailed information on the access rights of the different resources installed
- K8s core component infra assessment scan Kubernetes infra core components (etcd,apiserver,scheduler,controller-manager and etc) setting and configuration
- Compliance reports

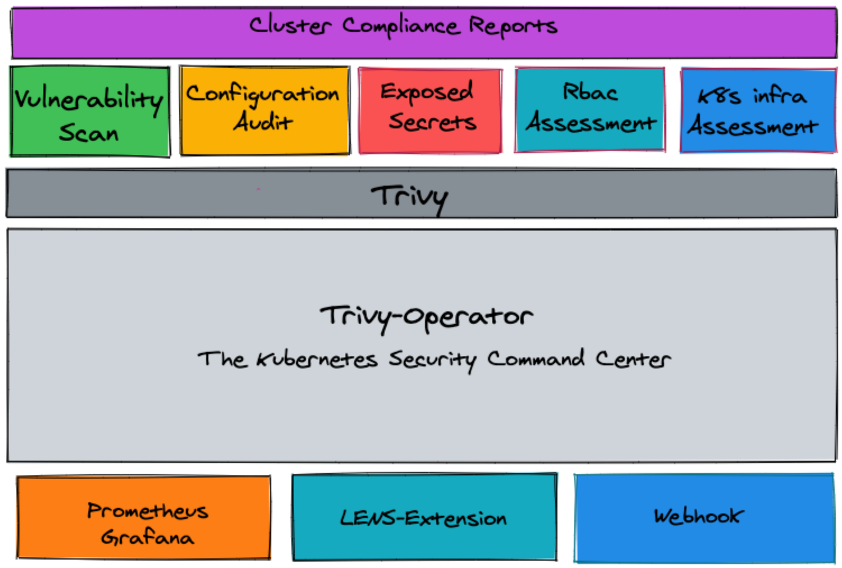

Figure 4.3: Trivy operator overview

Aqua posits that the Trivy Kubernetes operator, coupled with the Trivy cli tool lets them give full coverage on the development lifecycle.

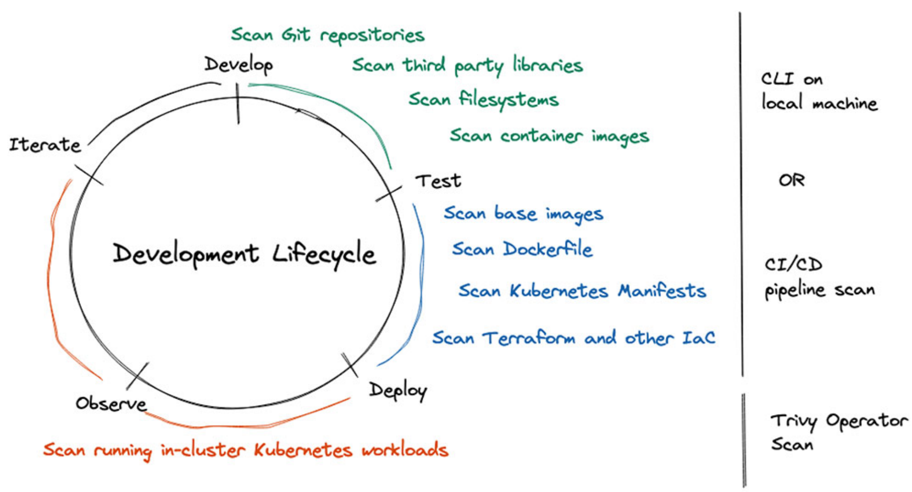

Figure 4.4: Trivy security scanning at different phases of your development lifecycle

While the development and test stages of the lifecycle are out of scope in this thesis, the fact that there is an operator tailored for Kubernetes clusters helps us to continuously assure the vulnerabilities of the images actually running in the cluster. The reports are stored as CRDs, but also have integrations with Prometheus metrics, OpenLens, webhooks, or Policy Reporter[60]. A report CRD looks as the following:


```yaml
apiVersion: aquasecurity.github.io/v1alpha1
kind: VulnerabilityReport
metadata:
  name: replicaset-nginx-6d4cf56db6-nginx
  namespace: default
  labels:
    trivy-operator.container.name: nginx
    trivy-operator.resource.kind: ReplicaSet
    trivy-operator.resource.name: nginx-6d4cf56db6
    trivy-operator.resource.namespace: default
    resource-spec-hash: 7cb64cb677
  uid: 8aa1a7cb-a319-4b93-850d-5a67827dfbbf
  ownerReferences:
    - apiVersion: apps/v1
      blockOwnerDeletion: false
      controller: true
      kind: ReplicaSet
      name: nginx-6d4cf56db6
      uid: aa345200-cf24-443a-8f11-ddb438ff8659
report:
  artifact:
    repository: library/nginx
    tag: '1.16'
  registry:
    server: index.docker.io
  scanner:
    name: Trivy
    vendor: Aqua Security
    version: 0.30.0
  summary:
    criticalCount: 2
    highCount: 0
    lowCount: 0
    mediumCount: 0
    unknownCount: 0
  vulnerabilities:
    - fixedVersion: 0.9.1-2+deb10u1
      installedVersion: 0.9.1-2
      links: []
      primaryLink: 'https://avd.aquasec.com/nvd/cve-2019-20367'
      resource: libbsd0
      score: 9.1
      severity: CRITICAL
      target: library/nginx:1.21.6
      title: ''
      vulnerabilityID: CVE-2019-20367
    - fixedVersion: ''
      installedVersion: 0.6.1-2
      links: []
```

```
primaryLink: 'https://avd.aquasec.com/nvd/cve-2018-25009'
resource: libwebp6
score: 9.1
severity: CRITICAL
target: library/nginx:1.16
title: 'libwebp: out-of-bounds read in WebPMuxCreateInternal'
vulnerabilityID: CVE-2018-25009
```

Listing 4.12: Trivy Operator Report

Open Policy Agent Gatekeeper

Figure 4.5: Open Policy Agent

Open Policy Agent (OPA) was created by Styra and a graduated project in the Cloud Native Computing Foundation (CNCF). It provides a high-level declarative language that lets you specify policy as code and simple PIS to offload policy decision-making from your software[61]. The language used to define the policies is called Rego[63] which extends Datalog[64] to support structured document models such as JSON. Rego queries are assertions on data stored in OPA. These queries can be used to define policies that enumerate instances of data that violate the expected state of the system.

OPA Gatekeeper is a Kubernetes Open Policy Agent implementation. Kubernetes allows decoupling policy decisions from the inner workings of the API Server by means of admission controller webhooks, which are executed whenever a resource is created, updated or deleted.

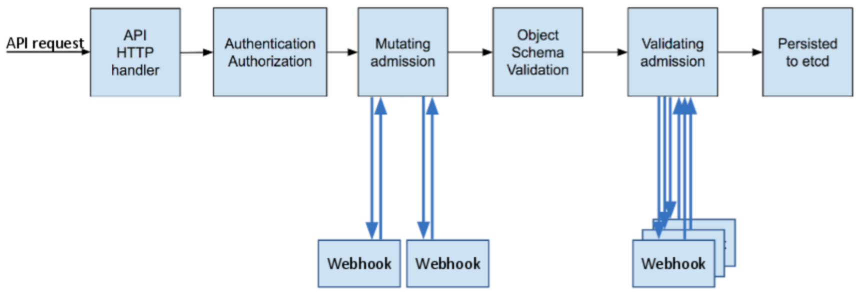

Figure 4.6: Kubernetes Admission Controller Phases[65]

Due to its integration with the Kubernetes admission controller webhooks, meaning it can interrupt and action before it is executed, Gatekeeper introduces the following functionality:

- An extensible, parameterized policy library
- Native Kubernetes CRDs for instantiating the policy library (aka "constraints")
- Native Kubernetes CRDs for extending the policy library (aka "constraint templates")
- Native Kubernetes CRDs for mutation support
- Audit functionality
- External data support

The way to enforce policies with OPA Gatekeeper is via two objects: Constraint Templates and Constraints. `ConstraintTemplate` objects describe the Rego that enforces the constraint and the schema of the constraint. The `Constraint` objects are used to inform Gatekeeper that the admin wants a `ConstraintTemplate` to be enforced and how[66]. As an example, this is how a constraint to enforce that objects have the "gatekeeper" Kubernetes label:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        # Schema for the `parameters` field
```

```
      openAPIV3Schema:
        type: object
        properties:
          labels:
            type: array
            items:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels

        violation[{"msg": msg, "details": {"missing_labels": missing}}] {
          provided := {label |input.review.object.metadata.labels[label]}
          required := {label |label := input.parameters.labels[_]}
          missing := required - provided
          count(missing) >0
          msg := sprintf("you must provide labels: %v", [missing])
        }
```

Listing 4.13: K8s Required Labels Constraint Template

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
  name: ns-must-have-gk
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Namespace"]
  parameters:
    labels: ["gatekeeper"]
```

Listing 4.14: K8s Required Labels Constraint

To find more examples of constraints the project maintains a Gatekeeper Library with use cases[67]. For the sake of clarity, only the `Constraint` objects will be specified for the following cases to show how and where should be applied. The `ConstraintTemplate` objects can be consulted in Annex A.

Gate images deployed to Kubernetes cluster

To accept only container images that begin with a string from a specified list:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: repo-is-openpolicyagent
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaces:
      - "default"
```

```
  parameters:
    repos:
      - "openpolicyagent/"
```

Listing 4.15: Repository constraints

Restrict exec commands on pods

To restrict exec commands from an interactive session, the operation to listen to is `CONNECT` for the resources `pods/exec` and `pods/attach`:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sDenyPodConnect
metadata:
  name: k8sdenypodconnect
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["pods/exec","pods/attach"]
  rules:
  - operations: ["CONNECT"]
EOF
```

Listing 4.16: Restrict exec commands constraint

Restrict over permissive containers

To control over permissive containers, we have to look to the `allowedCapabilities` and `requiredDropCapabilities` fields:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPCapabilities
metadata:
  name: capabilities-demo
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaces:
      - "default"
  parameters:
    allowedCapabilities: ["something"]
    requiredDropCapabilities: ["must_drop"]
```

Listing 4.17: Over permissive containers constraint

Restrict file and directory permissions

To control the usage of the host filesystem, the field to look at is `allowedHostPaths`:

```yaml
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPHostFilesystem
metadata:
  name: psp-host-filesystem
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  parameters:
    allowedHostPaths:
    - readOnly: true
      pathPrefix: "/foo"
```

Listing 4.18: File and directory permissions constraint

Ensure that pods meet defined Pod Security Standard

This constraint will specify to the `ConstraintTemplate` in which namespace to block containers with the `privileged` field:

```yaml
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPPrivilegedContainer
metadata:
  name: psp-privileged-container
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    excludedNamespaces: ["kube-system"]
```

Listing 4.19: Pod Security Standard constraint

Disable service account auto mount

Controls in which namespace to block pods with `automountServiceAccountTokenPod` enabled:

```yaml
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPAutomountServiceAccountTokenPod
metadata:
  name: psp-automount-serviceaccount-token-pod
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    excluded namespaces: ["kube-system"]
```

Listing 4.20: Automount constraint

Set requests and limits for containers

The following constraint specifies that all pods need to have defined `requests` and `limits` for CPU and memory resources:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredResources
metadata:
  name: container-must-have-limits-and-requests
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  parameters:
    limits:
      - cpu
      - memory
    requests:
      - cpu
      - memory
```

Listing 4.21: Container Resources constraint

Cilium and Tetragon

Figure 4.7: Cilium

Cilium provides network connectivity between applications deployed using Linux container management platforms like Docker and Kubernetes. At its core is the Linux kernel technology called eBPF, or extended Berkely Packet Filter, which enables the dynamic insertion of programming logic into the Linux kernel. Cilium is available as a commercially supported Kubernetes CNI plugin that can be used as an alternative to the AWS VPC CNI plugin on an Amazon EKS cluster.

Cilium functionality include[68]:

- **Protect and secure APIs transparently**: Traditional Kubernetes container network interfaces' firewall operate at Layer 3 and 4 of the Open Systesm Interconnection (OSI) model, but Cilium can also work at Layer 7 thus being able to filter network traffic at the application level

- **Secure service to service communication based on identities**: Cilium assigns a security identity to groups of application containers which share identical security policies. The identity is then associated with all network packets emitted by the application containers, allowing to validate the identity at the receiving node. Security identity management is performed using a key-value store

- **Secure access to and from external services**: Label based security is the tool of choice for cluster internal access control. In order to secure access to and from external services, traditional CIDR based security policies for both ingress and egress are supported. This allows to limit access to and from application containers to particular IP ranges

- **Simple Networking**: A simple flat Layer 3 network with the ability to span multiple clusters connects all application containers. IP allocation is kept simple by using host scope allocators. This means that each host can allocate IPs without any coordination between hosts.

- **Load Balancing**: Cilium implements distributed load balancing for traffic between application containers and to external services and is able to fully replace components such as kube-proxy. The load balancing is implemented in eBPF using efficient hashtables allowing for almost unlimited scale.

- **Bandwith Management**: Cilium implements bandwidth management through efficient EDT-based (Earliest Departure Time) rate-limiting with eBPF for container traffic that is egressing a node. This allows to significantly reduce transmission tail latencies for applications and to avoid locking under multi-queue NICs compared to traditional approaches such as HTB (Hierarchy Token Bucket) or TBF (Token Bucket Filter) as used in the bandwidth CNI plugin, for example.

Working with Cilium enables covering the following mitigations:

Use CNIs that are not prone to ARP poisoning

LB IPAM is a feature that allows Cilium to assign IP addresses to Services of type `LoadBalancer`. This functionality is usually left up to a cloud provider, however, when deploying in a private cloud environment, these facilities are not always available.

LB IPAM works in conjunction with features like the Cilium BGP Control Plane. Where LB IPAM is responsible for the allocation and assigning of IPs to Service objects and other features are responsible for load balancing and/or advertisement of these IPs.

How does Cilium help with ARP poisoning? If the source IP wasn't provided by Cilium's IPAM subsystem, we know it's a spoofed IP address and Cilium automatically blocks the traffic. Built-in Layer 3 Protection and IP Spoof Prevention are just some of the ways that Cilium automatically protects against common network attacks.

Network intrusion prevention

Cilium is both able to observe and enforce what behaviour happened inside of a Linux system. It can collect and filter out Security Observability data directly in the kernel and export it to user space as JSON events and / or store them in a specific log file via a Daemonset called hubble-enterprise. These JSON events are enriched with Kubernetes Identity Aware Information including services, labels, namespaces, pods and containers and with OS Level Process Visibility data including process binaries, pids, uids, parent binaries with the full Process Ancestry Tree. These events can then be exported in a variety of formats and sent to external systems such as a SIEM, e.g: Elasticsearch, Splunk or stored in an S3 bucket. For simplicity, in this blog post they will be directly consumed from the log file.

In the Isovalent blog the following use case is provided[69], where an attacker plans to use a *privileged pod* to try an reach the host namespace via a container escape. For this case we are not concerned right know how the breakout is performed, as there are multiple ways, but we want to detect that it has happened.

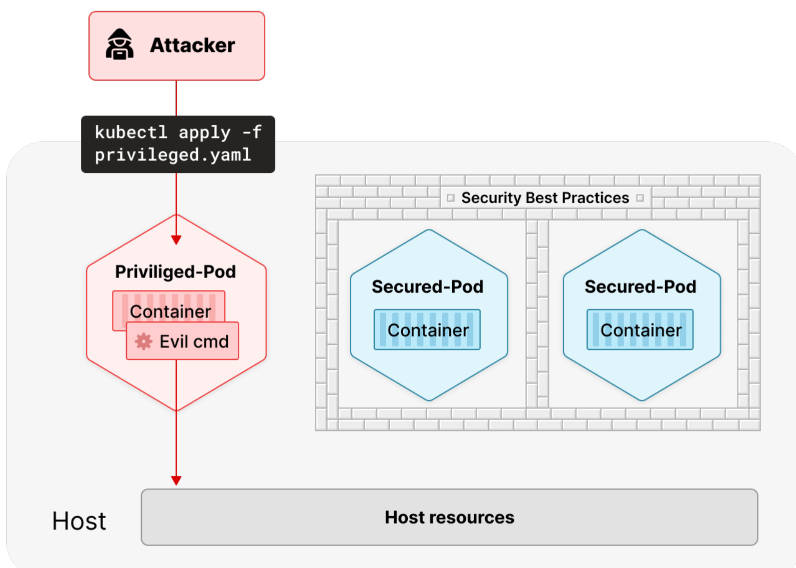

Figure 4.8: Container breakout

Cilium is capable to export the `process_exec` events in a JSON format, to be able to parse them easily. In the case of a breakout to the host from a privileged container called `privileged-the-pod`, we could see the following log where we can detect a container running in privileged mode(full logs can be inspected in Annex B) :

```
"process_exec":{
  "process":{
    "exec_id":"bWluaWt1YmU6MTEzNzkyNjAzMjk3MjoxNzk3OA==",
    "pid":17978,
    "uid":0,
    "cwd":"/",
    "binary":"/docker-entrypoint.sh",
    "arguments":"/docker-entrypoint.sh nginx -g \"daemon off;\"",
    "flags":"execve rootcwd clone",
    "start_time":"2021-10-13T12:58:31.794Z",
    "auid":4294967295,
    "pod":{
      "namespace":"default",
      "name":"privileged-the-pod",
      "container":{
        "id":"docker://32865
            cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
            ",
        "name":"privileged-the-pod",
        "image":{
          "id":"docker-pullable://nginxsha256:644
              a70516a26004c97d0d85c7fe1d0c3a67ea8ab7ddf4aff193d9f301670cf36
              ",
          "name":"nginx:latest"
        },
        "start_time":"2021-10-13T12:58:31Z"
      }
    },
    [...]
    "cap":{
      "permitted":[
        "CAP_CHOWN",
        "DAC_OVERRIDE",
        "CAP_DAC_READ_SEARCH",
        "CAP_FOWNER",
        "CAP_FSETID",
        "CAP_KILL",
        "CAP_SETGID",
        "CAP_SETUID",
        "CAP_SETPCAP",
        "CAP_LINUX_IMMUTABLE",
        "CAP_NET_BIND_SERVICE",
        "CAP_NET_BROADCAST",
        "CAP_NET_ADMIN",
        "CAP_NET_RAW",
        "CAP_IPC_LOCK",
        "CAP_IPC_OWNER",
        "CAP_SYS_MODULE",
        "CAP_SYS_RAWIO",
        "CAP_SYS_CHROOT",
        "CAP_SYS_PTRACE",
        "CAP_SYS_PACCT",
        "CAP_SYS_ADMIN",
        "CAP_SYS_BOOT",
```

```
        "CAP_SYS_NICE",
        "CAP_SYS_RESOURCE",
        "CAP_SYS_TIME",
        "CAP_SYS_TTY_CONFIG",
        "CAP_MKNOD",
        "CAP_LEASE",
        "CAP_AUDIT_WRITE",
        "CAP_AUDIT_CONTROL",
        "CAP_SETFCAP",
        "CAP_MAC_OVERRIDE",
        "CAP_MAC_ADMIN",
        "CAP_SYSLOG",
        "CAP_WAKE_ALARM",
        "CAP_BLOCK_SUSPEND",
        "CAP_AUDIT_READ"
      ],
```

Listing 4.22: Privileged container running

As a second step, the attacker can use kubectl exec to get shell access to `privileged-the-pod`:

```
 "process_exec":{
    "process":{
      "exec_id":"bWluaWt1YmU6MTI5NDM3OTU0NzQ3ODoxOTU5NA==",
      "pid":19594,
      "uid":0,
      "cwd":"/",
      "binary":"/bin/bash",
      "flags":"execve rootcwd clone",
      "start_time":"2021-10-13T13:01:08.248Z",
      "auid":4294967295,
      "pod":{
        "namespace":"default",
        "name":"privileged-the-pod",
        "container":{
          "id":"docker://32865
              cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
              ",
          "name":"privileged-the-pod",
          "image":{
            "id":"docker-pullable://nginxsha256:644
                a70516a26004c97d0d85c7fe1d0c3a67ea8ab7ddf4aff193d9f301670cf36
                ",
            "name":"nginx:latest"
          },
          "start_time":"2021-10-13T12:58:31Z"
        }
      },
```

Listing 4.23: Container executing a shell

Finally the attacker enters the host via `nsenter` command:

```
  "process_exec":{
    "process":{
      "exec_id":"bWluaWt1YmU6MTYyNzIwMjkzMjkyMToyMzc0Ng==",
      "pid":23746,
```

```
"uid":0,
"cwd":"/",
"binary":"/usr/bin/nsenter",
"arguments":"-t 1 -a bash",
"flags":"execve rootcwd clone",
"start_time":"2021-10-13T13:06:41.071Z",
"auid":4294967295,
"pod":{
  "namespace":"default",
  "name":"privileged-the-pod",
  "container":{
    "id":"docker://32865
        cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
        ",
    "name":"privileged-the-pod",
    "image":{
      "id":"docker-pullable://nginxsha256:644
          a70516a26004c97d0d85c7fe1d0c3a67ea8ab7ddf4aff193d9f301670cf36
          ",
      "name":"nginx:latest"
    },
    "start_time":"2021-10-13T12:58:31Z"
  }
},
```

Listing 4.24: Container entering host via nsenter command

The above example shows that with Cilium we can export the necessary events to be aware of attacker intrussions and act accordingly.

Restrict container runtime using LSM

While the mitigation specifically calls out for LSM, in reality there are several options to restrict container runtime[70]:

- Application/system level
  - App Instrumentation
  - LD_PRELOAD
  - ptrace(2)
- Kernel level
  - Secure computing mode (seccomp)
  - SELinux/LSM
  - Kernel Module
  - Tetragon + eBPF

This is a complex subject that merits its own chapter but unfortunately it falls out of scope of this Thesis, so in reality the choice of using Tetragon and eBPF answers to three key benefits:

- **Application transparency**: the developers do not have to worry about including any type of instrumentation
- **Extendibility**: while having the benefits of working at a kernel level, custom modules can be developed without the security and availability risks
- **Synchronous enforcement**: via Cilium tracing policies, actions can be defined to react to events

The first step to react to an event is to be aware of it via a Cilium Tracing Policy. A policy allows users to trace arbitrary events in the kernel and optionally define actions to take on a match for enforcement. At the moment of this writing, the actions available for Tetragon are:

- Sigkill action
- Signal action
- Override action
- FollowFD action
- UnfollowFD action
- CopyFD action
- GetUrl action
- DnsLookup action
- Post action
- NoPost action

In the following example, we can see how we can prevent writing to `/etc/passwd` with a `Sigkill` action:

```
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: "syswritefollowfdpsswd"
spec:
  kprobes:
  - call: "fd_install"
    syscall: false
    args:
    - index: 0
      type: int
    - index: 1
      type: "file"
    selectors:
    - matchPIDs:
      - operator: NotIn
        followForks: true
```

```
        isNamespacePID: true
        values:
        - 0
        - 1
      matchArgs:
      - index: 1
        operator: "Equal"
        values:
        - "/etc/passwd"
      matchActions:
      - action: FollowFD
        argFd: 0
        argName: 1
  - call: "sys_close"
    syscall: true
    args:
    - index: 0
      type: "int"
    selectors:
    - matchPIDs:
      - operator: NotIn
        followForks: true
        isNamespacePID: true
        values:
        - 0
        - 1
      matchActions:
      - action: UnfollowFD
        argFd: 0
        argName: 0
  - call: "sys_write"
    syscall: true
    args:
    - index: 0
      type: "fd"
    - index: 1
      type: "char_buf"
      sizeArgIndex: 3
    - index: 2
      type: "size_t"
    selectors:
    - matchPIDs:
      - operator: NotIn
        followForks: true
        isNamespacePID: true
        values:
        - 0
        - 1
      matchArgs:
      - index: 0
        operator: "Prefix"
        values:
        - "/etc/passwd"
      matchActions:
      - action: Sigkill
```

Listing 4.25: Prevent writing to /etc/passwd

As there is no "one fits all" solution in terms of defending assets. Each case should be studied, prioritized, and then protected via a `tracing policy` with an `action` attached. Annex C contains a few Cilium Tracing Policies that can be found in the Tetragon repository[71].

Custom solutions

Remove unused secrets from the cluster

There are few solutions that offer an approach to this problem, and fewer are open-source. Identifying unused secrets within a cluster can be challenging since Kubernetes secrets do not retain information about their usage. This means that to get the list of secrets we have to rely on comparing the list of existing secrets with references in Kubernetes objects. Secrets can be referenced in:

- Ingresses TLS secrets
- Pods spec:
  - Environment secrets
  - Volumes secrets
  - ImagePullSecrets
- Custom Resource definitions

We could generate the list of secrets in these resources and compare it with the list of existing secrets:

```
envSecrets=$(kubectl get pods -o jsonpath='{.items[*].spec.containers
    [*].env[*].valueFrom.secretKeyRef.name}' | xargs -n1)
envSecrets2=$(kubectl get pods -o jsonpath='{.items[*].spec.containers
    [*].envFrom[*].secretRef.name}' | xargs -n1)
volumeSecrets=$(kubectl get pods -o jsonpath='{.items[*].spec.volumes
    [*].secret.secretName}' | xargs -n1)
pullSecrets=$(kubectl get pods -o jsonpath='{.items[*].spec.
    imagePullSecrets[*].name}' | xargs -n1)
tlsSecrets=$(kubectl get ingress -o jsonpath='{.items[*].spec.tls[*].
    secretName}' | xargs -n1)

diff \
<(echo "$envSecrets\n$envSecrets2\n$volumeSecrets\n$pullSecrets\
    n$tlsSecrets" | sort | uniq) \
<(kubectl get secrets -o jsonpath='{.items[*].metadata.name}' | xargs -
    n1 | sort | uniq)
```

Listing 4.26: Getting and Deleting Orphaned Secrets with Kubectl[72]

There are other implementations out there (for example in Go[73], without taking into account the TLS certificates in the ingresses), but this approach leaves out the CRD objects, as their structure is not standard, so depending on the CRDs deployed in the cluster the script should be modified to take into account possible secret references, as we could potentially delete a secret in use otherwise. This means that at the moment, the way Kubernetes Secrets work, the only way to detect unused secrets is to parse every object in the cluster for secret references. And again, this would mean having prior knowledge of every custom object deployed in our cluster.

Chapter 5

Conclussions and further work

5.1 Plan of action

As stated in the previous chapter the first priority should be to reduce the attack surface and for that the first step would be to *Limit access to services over network* so that the only point exposed to the internet is the webpage that the users interact with. After that, the Kubernetes API should be firewalled. For this the recomendation is to migrate to EKS with the cluster endpoint set to private access instead of using a manually deployed Kubernetes cluster in EC2 virtual machines. The change to EKS will also automatically cover the following mitigations:

- *Restrict access to etcd*
- *Restrict the usage of unauthenticated APIs in the cluster*
- *Collect logs to remote data storage*

When creating the new cluster, Cilium should be configured as the CNI to use, which will open the door to other mitigations later on. But while the change is being made, AWS provides a way to limit access to the Kubernetes API residing in the EC2 virtual machines via Security Groups.

Following that, the recommendation would be to work on user permissions and implement mitigations as *Adhere to least-privilege principle*, *Restrict exec commands on pods* (via RBAC permissions) and *Multi-factor authentication*.

The next step should be improving the observability of our cluster, and for that the Trivy Operator will give reports of vulnerabilities of the images containers running in the cluster as well as potential Kubernetes misconfigurations.

After that, it depends on the technical level of the cluster administrators, as using OPA and Tetragon without technical know-how would be the same as applying best practices from a Hardening Guide or Kubernetes benchmark. It won't hurt to have some controls in place, but they could not really make sense for the current scenario and result in time badly spent. Kubernetes training for the employees administrating the cluster is highly recommended as an investment to be able to apply the controls that make sense. Once the desired technical level is achieved, OPA Gatekeeper can help enforce policies, and Cilium and Tetragon can help with *Network intrusion prevention* and real-time reaction via eBPF hooks.

Once all those mitigations are implemented, the team can focus on other fields like defense in depth with protections such as *Use NodeRestriction admission controller*, *Allocate specific identities to pods* and *Use managed secret store.* Also, quality of life features like *Enable Just In Time access to API server*, *Use cloud storage provider and Implement data backup strategy* and *Remove unused secrets from the cluster.*

5.2 Next steps

By no means this Master's Thesis goal was to provide an exhaustive, definitive, one-shot protection guide. The main objective has been to create a stepping stone for each attack vector on which to begin to protect a Kubernetes cluster in a Cloud environment. The solutions here presented are to be considered as a work-in-progress from which to iterate, giving value from the first moment, and growing from there as the technical know-how of the employees improves.

There are entire fields that have not been broached for not aligning exactly with one of the mitigations defined by Microsoft in its threat matrix, but that does not mean they are not important. From separating workloads with network policies, to being able to handle denial of service attacks, to legal requirements, to supply chain concerns or more in-depth protection against insider threats, there is always a way to improve your security posture. Moreover, as technology advances new threats arise, so it is equally important to keep track of the current cybersecurity threats. Ultimately, cybersecurity boils down to a constant race against the ever-evolving advancements made by threat actors, given the rapidly changing nature of the field.

List of Figures

List of Tables

Listings

Bibliography

[1] 2022 Official Cybercrime Report. eSentire. Retrieved on the 11th of March, 2023. https://www.esentire.com/resources/library/2022-official-cybercrime-report#:~:text=According%20to%20Cybersecurity%20Ventures%2C%20the%20global%20annual%20cost,is%20expected%20to%20reach%20%2410.5%20trillion%20by%202025.

[2] The Global Risks Report 2022. World Economic Forum. Retrieved on the 10th of March, 2023 https://www3.weforum.org/docs/WEF_The_Global_Risks_Report_2022.pdf

[3] State of cybersecurity 2022. CompTIA. Retrieved on the 9th of March, 2023 https://www.comptia.org/content/research/cybersecurity-trends-research

[4] 2022 state of Kubernetes security report. RedHat. Retrieved on the 9th of March, 2023. https://www.redhat.com/en/resources/state-kubernetes-security-report

[5] The State of Cloud Security Report 2022. Snyk. Retrieved on the 13th of March, 2023. https://snyk.io/reports/state-of-cloud-security/

[6] The State of Cloud-Native Security 2023 Report. Palo Alto Networks. Retrieved on the 13th of March, 2023. https://www.paloaltonetworks.com/state-of-cloud-native-security

[7] The state of Kubernetes {Open-Source} Security. Armo. Retrieved on the 12th of March, 2023. https://landing.armosec.io/state-of-kubernetes-open-source-security-2022

[8] Gartner® Report: Top Trends in Cybersecurity 2022. Retrieved on the 10th of March, 2023 https://www.gartner.com/doc/reprints?id=1-29OTFFPI&ct=220411&st=sb

[9] Center for Internet Security. *Kubernetes Benchmark* https://www.cisecurity.org/benchmark/kubernetes

[10] National Security Agency. *Kubernetes Hardening Guide* https://media.defense.gov/2022/Aug/29/2003066362/-1/-1/0/CTR_KUBERNETES_HARDENING_GUIDANCE_1.2_20220829.PDF

[11] Defense Information Systems Agency. *Kubernetes Security Implementation Guide* https://public.cyber.mil/stigs/downloads/

[12] Amazon Web Services: EKS Best Practices Guide for Security. Retrieved on the 21st of March, 2023 https://aws.github.io/aws-eks-best-practices/security/docs/

[13] Google Cloud Platform: Harden your cluster's security. Retrieved on the 27th of March, 2023 https://cloud.google.com/kubernetes-engine/docs/how-to/hardening-your-cluster

[14] Azure Cloud: Best practices for cluster security and upgrades in Azure Kubernetes Service (AKS). Retrieved on the 27th of March, 2023 https://learn.microsoft.com/en-us/azure/aks/operator-best-practices-cluster-security?tabs=azure-cli

[15] Too much to Choose - Making Sense of a Smorgasboard of Security Standard. Anais Urlichs & Rory McCune. https://www.youtube.com/watch?v=yKqqCxvlDeE

[16] Sig-Security K8s Threat Model. Retrieved on the 12th of March, 2023. https://github.com/cncf/financial-user-group/tree/main/projects/k8s-threat-model

[17] Threat matrix for Kubernetes. Retrieved on the 13th of March, 2023 https://www.microsoft.com/en-us/security/blog/2020/04/02/attack-matrix-kubernetes/

[18] Secure containerized environments with updated threat matrix for Kubernetes. Retrieved on the 13th of March, 2023 https://www.microsoft.com/en-us/security/blog/2021/03/23/secure-containerized-environments-with-updated-threat-matrix-for-kubernetes/

[19] Microsoft Threat Matrix for Kubernetes. Retrieved on the 12th of March, 2023. https://microsoft.github.io/Threat-Matrix-for-Kubernetes/

[20] MITRE ATT&CK Matrix for Entreprise. Retrieved on the 22nd of March, 2023. https://attack.mitre.org/

[21] Martin, Andrew. Hausenblas, Michael. *Hacking Kubernetes.* O'Reilly Media, Inc. October 2021.

[22] Salazar, Jed. Reka Ivanko, Natalia. *Security Observability with eBPF.* O'Reilly Media, Inc. 2022

[23] DEX. Retrieved on the 4th of April, 2023 https://dexidp.io/

[24] Pinniped. Retrieved on the 4th of April, 2023 https://pinniped.dev/

[25] VMware Tanzu. Retrieved on the 4th of April, 2023 https://tanzu.vmware.com/tanzu

[26] Cluster authentication. Amazon Elastic Kubernetes Services. Retrieved on the 4th of April, 2023 https://docs.aws.amazon.com/eks/latest/userguide/cluster-auth.html

[27] Authenticating to the Kubernetes API server. Google Kubernetes Engine. Retrieved on the 4th of April, 2023 https://cloud.google.com/kubernetes-engine/docs/how-to/api-server-authentication

[28] Access and identity options for Azure Kubernetes Service. Retrieved on the 4th of April, 2023 https://learn.microsoft.com/en-us/azure/aks/concepts-identity

[29] The all-in-one open source security scanner. Retrieved on the 5th of April, 2023 https://trivy.dev/

[30] Open Policy Agent Gatekeeper. Retrieved on the 5th of April, 2023 https://github.com/open-policy-agent/gatekeeper

[31] eBPF Foundation. Retrieved on the 20th of May, 2023 https://ebpf.foundation/

[32] Falco, open source standard for runtime security for hosts, containers, Kubernetes and the cloud. Retrieved on the 5th of April, 2023 https://falco.org/

[33] Tetragon, eBPF-based Security and Runtime Enforcement. Retrieved on the 5th of April, 2023 https://github.com/cilium/tetragon

[34] Using RBAC Authorization. Retrieved on the 17th of April, 2023 https://kubernetes.io/docs/reference/access-authn-authz/rbac/

[35] NodeRestriction Admission Controller. Retrieved on the 17th of April, 2023 https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#noderestriction

[36] Customizing kubelet configuration. Retrieved on the 10th of May, 2023 https://eksctl.io/usage/customizing-the-kubelet/

[37] Eksctl Config file schema. Retrieved on the 10th of May, 2023 https://eksctl.io/usage/schema/

[38] Multi-Factor Authentication (MFA) for IAM. Retrieved on the 12th of May, 2023 https://aws.amazon.com/es/iam/features/mfa/

[39] Enabling a FIDO security key (console). Retrieved on the 12th of May, 2023 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_enable_fido.html

[40] Enabling a virtual multi-factor authentication (MFA) device (console). Retrieved on the 12th of May, 2023 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_enable_virtual.html

[41] Enabling a hardware TOTP token (console). Retrieved on the 12th of May, 2023 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_enable_physical.html

[42] Amazon EKS cluster endpoint access control. Retrieved on the 25th of April, 2023 https://docs.aws.amazon.com/eks/latest/userguide/cluster-endpoint.html

[43] A minimal reference implementation for temporary elevated access. Retrieved on the 12th of May, 2023 https://github.com/aws-samples/aws-iam-temporary-elevated-access-broker

[44] Network load balancing on Amazon EKS. Retrieved on the 14th of May, 2023 https://docs.aws.amazon.com/eks/latest/userguide/network-load-balancing.html

[45] AWS Network Load Balancer annotations. Retrieved on the 14th of May, 2023 https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.2/guide/service/annotations

[46] Restrict the use of host networking and block access to instance metadata service. Retrieved on the 15th of May, 2023 https://docs.aws.amazon.com/whitepapers/latest/security-practices-multi-tenant-saas-applications-eks/restrict-the-use-of-host-networking-and-block-access-to-instance-metadata-service.html

[47] Cilium Layer 3 Network Policies. Retrieved on the 15th of May, 2023 https://docs.cilium.io/en/latest/security/policy/language/

[48] IAM Roles for Service Accounts (IRSA). Retrieved on the 18th of May, 2023 https://aws.github.io/aws-eks-best-practices/security/docs/iam/#iam-roles-for-service-accounts-irsa

[49] Configuring a Kubernetes service account to assume an IAM role. Retrieved on the 18th of May, 2023 https://docs.aws.amazon.com/eks/latest/userguide/associate-service-account-role.html

[50] Using AWS Secrets Manager secrets with Kubernetes. Retrieved on the 18th of May, 2023 https://docs.aws.amazon.com/eks/latest/userguide/manage-secrets.html

[51] Use AWS Secrets Manager secrets in Amazon Elastic Kubernetes Service. Retrieved on the 18th of May, 2023 https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_csi_driver.html

[52] Amazon EBS CSI driver. Retrieved on the 2nd of May, 2023 https://docs.aws.amazon.com/eks/latest/userguide/ebs-csi.html

[53] Amazon EBS snapshots. Retrieved on the 2nd of May, 2023 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSSnapshots.html

[54] Amazon EKS control plane logging. Retrieved on the 3rd of May, 2023 https://docs.aws.amazon.com/eks/latest/userguide/control-plane-logs.html

[55] What is Amazon CloudWatch?. Retrieved on the 3rd of May, 2023 https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html

[56] Quick Start setup for Container Insights on Amazon EKS and Kubernetes. Retrieved on the 3rd of May, 2023 https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Container-Insights-setup-EKS-quickstart.html

[57] Trivy GitHub repository. Retrieved on the 5th of May, 2023 https://github.com/aquasecurity/trivy

[58] Kubernetes controllers. Retrieved on the 5th of May, 2023 https://kubernetes.io/docs/concepts/architecture/controller/

[59] Trivy Operator GitHub repository. Retrieved on the 5th of May, 2023 https://github.com/aquasecurity/trivy-operator

[60] Trivy Policy Reporter integration. Retrieved on the 5th of May, 2023 https://aquasecurity.github.io/trivy-operator/v0.14.0/tutorials/integrations/policy-reporter/

[61] Open Policy Agent. Retrieved on the 8th of May, 2023 https://www.openpolicyagent.org/docs/latest/

[62] Open Policy Agent Gatekeeper. Retrieved on the 8th of May, 2023 https://open-policy-agent.github.io/gatekeeper/website/docs/

[63] Rego language. Retrieved on the 8th of May, 2023 https://www.openpolicyagent.org/docs/latest/policy-language/

[64] Datalog language. Retrieved on the 8th of May, 2023 https://en.wikipedia.org/wiki/Datalog

[65] A Guide to Kubernetes Admission Controllers. Retrieved on the 12th of May, 2023 https://kubernetes.io/blog/2019/03/21/a-guide-to-kubernetes-admission-controllers/

[66] How to use Gatekeeper. Retrieved on the 16th of May, 2023 https://open-policy-agent.github.io/gatekeeper/website/docs/howto

[67] OPA Gatekeeper Library. Retrieved on the 16th of May, 2023 https://open-policy-agent.github.io/gatekeeper-library/

[68] Cilium functionality overview Retrieved on the 8th of May, 2023 https://docs.cilium.io/en/stable/overview/intro/#functionality-overview

[69] Detecting a Container Escape with Cilium and eBPF Retrieved on the 10th of May, 2023 https://isovalent.com/blog/post/2021-11-container-escape/

[70] Tetragon – eBPF-based Security Observability & Runtime Enforcement. Retrieved on the 7th of May, 2023 https://isovalent.com/blog/post/2022-05-16-tetragon/

[71] Tetragon tracing policies. Retrieved on the 7th of May. 2023 https://github.com/cilium/tetragon/blob/main/docs/content/en/docs/reference/tracing-policy.md

[72] Getting and Deleting Orphaned Secrets with Kubectl. Retrieved on the 7th of May, 2023 https://www.blinkops.com/blog/getting-and-deleting-orphaned-secrets-with-kubectl

[73] k8s-unused-secret-detector. Retrieved on the 10th of May, 2023 https://github.com/dtan4/k8s-unused-secret-detector

Appendix A

Open Policy Agent Gatekeeper Constraint Templates

`Constraint Templates` for the `Constraints` defined in section 4.2.3.

A.1 Gate images deployed to Kubernetes cluster

```yaml
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sallowedrepos
  annotations:
    metadata.gatekeeper.sh/title: "Allowed Repositories"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Requires container images to begin with a string from the specified
          list.
spec:
  crd:
    spec:
      names:
        kind: K8sAllowedRepos
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          properties:
            repos:
              description: The list of prefixes a container image is
                  allowed to have.
              type: array
```

```
          items:
            type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8sallowedrepos

        violation[{"msg": msg}] {
          container := input.review.object.spec.containers[_]
          satisfied := [good |repo = input.parameters.repos[_] ; good =
              startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>,
               allowed repos are %v", [container.name, container.image,
              input.parameters.repos])
        }

        violation[{"msg": msg}] {
          container := input.review.object.spec.initContainers[_]
          satisfied := [good |repo = input.parameters.repos[_] ; good =
              startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("initContainer <%v> has an invalid image repo <%
              v>, allowed repos are %v", [container.name, container.image
              , input.parameters.repos])
        }

        violation[{"msg": msg}] {
          container := input.review.object.spec.ephemeralContainers[_]
          satisfied := [good |repo = input.parameters.repos[_] ; good =
              startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("ephemeralContainer <%v> has an invalid image
              repo <%v>, allowed repos are %v", [container.name,
              container.image, input.parameters.repos])
        }
```

Listing A.1: Gate images deployed to Kubernetes cluster

A.2 Restrict exec commands

```
apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
metadata:
  name: k8sdenypodconnect
spec:
  crd:
    spec:
      names:
        kind: K8sDenyPodConnect
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8sdenypodconnect
        violation[{"msg": msg}] {
```

```
        msg := sprintf("REVIEW OBJECT: %v", [input.review])
      }
EOF
```

Listing A.2: Restrict exec command constraint template

A.3 Restrict over permissive containers

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spspcapabilities
  annotations:
    metadata.gatekeeper.sh/title: "Capabilities"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Controls Linux capabilities on containers. Corresponds to the
      `allowedCapabilities` and `requiredDropCapabilities` fields in a
      PodSecurityPolicy. For more information, see
      https://kubernetes.io/docs/concepts/policy/pod-security-policy/#
          capabilities
spec:
  crd:
    spec:
      names:
        kind: K8sPSPCapabilities
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          description: >-
            Controls Linux capabilities on containers. Corresponds to the
            `allowedCapabilities` and `requiredDropCapabilities` fields in
                 a
            PodSecurityPolicy. For more information, see
            https://kubernetes.io/docs/concepts/policy/pod-security-policy
                 /#capabilities
          properties:
            exemptImages:
              description: >-
                Any container that uses an image that matches an entry in
                     this list will be excluded
                from enforcement. Prefix-matching can be signified with `*
                     `. For example: `my-image-*`.

                It is recommended that users use the fully-qualified Docker
                      image name (e.g. start with a domain name)
                in order to avoid unexpectedly exempting images from an
                     untrusted repository.
              type: array
              items:
                type: string
            allowedCapabilities:
              type: array
```

```
          description: "A list of Linux capabilities that can be added
              to a container."
          items:
            type: string
        requiredDropCapabilities:
          type: array
          description: "A list of Linux capabilities that are required
              to be dropped from a container."
          items:
            type: string
targets:
  - target: admission.k8s.gatekeeper.sh
    rego: |
      package capabilities

      import data.lib.exempt_container.is_exempt

      violation[{"msg": msg}] {
        container := input.review.object.spec.containers[_]
        not is_exempt(container)
        has_disallowed_capabilities(container)
        msg := sprintf("container <%v> has a disallowed capability.
            Allowed capabilities are %v", [container.name, get_default(
            input.parameters, "allowedCapabilities", "NONE")])
      }

      violation[{"msg": msg}] {
        container := input.review.object.spec.containers[_]
        not is_exempt(container)
        missing_drop_capabilities(container)
        msg := sprintf("container <%v> is not dropping all required
            capabilities. Container must drop all of %v or \"ALL\"", [
            container.name, input.parameters.requiredDropCapabilities])
      }



      violation[{"msg": msg}] {
        container := input.review.object.spec.initContainers[_]
        not is_exempt(container)
        has_disallowed_capabilities(container)
        msg := sprintf("init container <%v> has a disallowed capability
            . Allowed capabilities are %v", [container.name, get_default
            (input.parameters, "allowedCapabilities", "NONE")])
      }

      violation[{"msg": msg}] {
        container := input.review.object.spec.initContainers[_]
        not is_exempt(container)
        missing_drop_capabilities(container)
        msg := sprintf("init container <%v> is not dropping all
            required capabilities. Container must drop all of %v or \"
            ALL\"", [container.name, input.parameters.
            requiredDropCapabilities])
      }
```

```
    violation[{"msg": msg}] {
      container := input.review.object.spec.ephemeralContainers[_]
      not is_exempt(container)
      has_disallowed_capabilities(container)
      msg := sprintf("ephemeral container <%v> has a disallowed
          capability. Allowed capabilities are %v", [container.name,
          get_default(input.parameters, "allowedCapabilities", "NONE")
          ])
    }

    violation[{"msg": msg}] {
      container := input.review.object.spec.ephemeralContainers[_]
      not is_exempt(container)
      missing_drop_capabilities(container)
      msg := sprintf("ephemeral container <%v> is not dropping all
          required capabilities. Container must drop all of %v or \"
          ALL\"", [container.name, input.parameters.
          requiredDropCapabilities])
    }


    has_disallowed_capabilities(container) {
      allowed := {c |c := lower(input.parameters.allowedCapabilities[_
          ])}
      not allowed["*"]
      capabilities := {c |c := lower(container.securityContext.
          capabilities.add[_])}

      count(capabilities - allowed) >0
    }

    missing_drop_capabilities(container) {
      must_drop := {c |c := lower(input.parameters.
          requiredDropCapabilities[_])}
      all := {"all"}
      dropped := {c |c := lower(container.securityContext.capabilities
          .drop[_])}

      count(must_drop - dropped) >0
      count(all - dropped) >0
    }

    get_default(obj, param, _default) = out {
      out = obj[param]
    }

    get_default(obj, param, _default) = out {
      not obj[param]
      not obj[param] == false
      out = _default
    }
  libs:
    - |
      package lib.exempt_container

      is_exempt(container) {
```

```
      exempt_images := object.get(object.get(input, "parameters",
          {}), "exemptImages", [])
      img := container.image
      exemption := exempt_images[_]
      _matches_exemption(img, exemption)
    }

    _matches_exemption(img, exemption) {
      not endswith(exemption, "*")
      exemption == img
    }

    _matches_exemption(img, exemption) {
      endswith(exemption, "*")
      prefix := trim_suffix(exemption, "*")
      startswith(img, prefix)
    }
```

Listing A.3: Restrict over permissive containers constraint template

A.4 Restrict file and directory permissions

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spsphostfilesystem
  annotations:
    metadata.gatekeeper.sh/title: "Host Filesystem"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Controls usage of the host filesystem. Corresponds to the
      `allowedHostPaths` field in a PodSecurityPolicy. For more
          information,
      see
      https://kubernetes.io/docs/concepts/policy/pod-security-policy/#
          volumes-and-file-systems
spec:
  crd:
    spec:
      names:
        kind: K8sPSPHostFilesystem
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          description: >-
            Controls usage of the host filesystem. Corresponds to the
            `allowedHostPaths` field in a PodSecurityPolicy. For more
                information,
            see
            https://kubernetes.io/docs/concepts/policy/pod-security-policy
                /#volumes-and-file-systems
          properties:
            allowedHostPaths:
              type: array
```

```
          description: "An array of hostpath objects, representing
              paths and read/write configuration."
          items:
            type: object
            properties:
              pathPrefix:
                type: string
                description: "The path prefix that the host volume must
                    match."
              readOnly:
                type: boolean
                description: "when set to true, any container
                    volumeMounts matching the pathPrefix must include `
                    readOnly: true`."
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8spsphostfilesystem

        violation[{"msg": msg, "details": {}}] {
            volume := input_hostpath_volumes[_]
            allowedPaths := get_allowed_paths(input)
            input_hostpath_violation(allowedPaths, volume)
            msg := sprintf("HostPath volume %v is not allowed, pod: %v.
                Allowed path: %v", [volume, input.review.object.metadata.
                name, allowedPaths])
        }

        input_hostpath_violation(allowedPaths, volume) {
            # An empty list means all host paths are blocked
            allowedPaths == []
        }
        input_hostpath_violation(allowedPaths, volume) {
            not input_hostpath_allowed(allowedPaths, volume)
        }

        get_allowed_paths(arg) = out {
            not arg.parameters
            out = []
        }
        get_allowed_paths(arg) = out {
            not arg.parameters.allowedHostPaths
            out = []
        }
        get_allowed_paths(arg) = out {
            out = arg.parameters.allowedHostPaths
        }

        input_hostpath_allowed(allowedPaths, volume) {
            allowedHostPath := allowedPaths[_]
            path_matches(allowedHostPath.pathPrefix, volume.hostPath.path)
            not allowedHostPath.readOnly == true
        }

        input_hostpath_allowed(allowedPaths, volume) {
            allowedHostPath := allowedPaths[_]
            path_matches(allowedHostPath.pathPrefix, volume.hostPath.path)
```

```
    allowedHostPath.readOnly
    not writeable_input_volume_mounts(volume.name)
}

writeable_input_volume_mounts(volume_name) {
    container := input_containers[_]
    mount := container.volumeMounts[_]
    mount.name == volume_name
    not mount.readOnly
}

# This allows "/foo", "/foo/", "/foo/bar" etc., but
# disallows "/fool", "/etc/foo" etc.
path_matches(prefix, path) {
    a := path_array(prefix)
    b := path_array(path)
    prefix_matches(a, b)
}
path_array(p) = out {
    p != "/"
    out := split(trim(p, "/"), "/")
}
# This handles the special case for "/", since
# split(trim("/", "/"), "/") == [""]
path_array("/") = []

prefix_matches(a, b) {
    count(a) <= count(b)
    not any_not_equal_upto(a, b, count(a))
}

any_not_equal_upto(a, b, n) {
    a[i] != b[i]
    i < n
}

input_hostpath_volumes[v] {
    v := input.review.object.spec.volumes[_]
    has_field(v, "hostPath")
}

# has_field returns whether an object has a field
has_field(object, field) = true {
    object[field]
}
input_containers[c] {
    c := input.review.object.spec.containers[_]
}

input_containers[c] {
    c := input.review.object.spec.initContainers[_]
}

input_containers[c] {
    c := input.review.object.spec.ephemeralContainers[_]
}
```

Listing A.4: Restrict file and directory permissions Constraint Template

A.5 Ensure that pods meet defined Pod Security Standard

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spspprivilegedcontainer
  annotations:
    metadata.gatekeeper.sh/title: "Privileged Container"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Controls the ability of any container to enable privileged mode.
      Corresponds to the `privileged` field in a PodSecurityPolicy. For
           more
      information, see
      https://kubernetes.io/docs/concepts/policy/pod-security-policy/#
           privileged
spec:
  crd:
    spec:
      names:
        kind: K8sPSPPrivilegedContainer
      validation:
        openAPIV3Schema:
          type: object
          description: >-
            Controls the ability of any container to enable privileged
                 mode.
            Corresponds to the `privileged` field in a PodSecurityPolicy.
                 For more
            information, see
            https://kubernetes.io/docs/concepts/policy/pod-security-policy
                 /#privileged
          properties:
            exemptImages:
              description: >-
                Any container that uses an image that matches an entry in
                     this list will be excluded
                from enforcement. Prefix-matching can be signified with `*
                     `. For example: `my-image-*`.

                It is recommended that users use the fully-qualified Docker
                      image name (e.g. start with a domain name)
                in order to avoid unexpectedly exempting images from an
                     untrusted repository.
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
```

```
    rego: |
      package k8spspprivileged

      import data.lib.exempt_container.is_exempt

      violation[{"msg": msg, "details": {}}] {
          c := input_containers[_]
          not is_exempt(c)
          c.securityContext.privileged
          msg := sprintf("Privileged container is not allowed: %v,
              securityContext: %v", [c.name, c.securityContext])
      }

      input_containers[c] {
          c := input.review.object.spec.containers[_]
      }

      input_containers[c] {
          c := input.review.object.spec.initContainers[_]
      }

      input_containers[c] {
          c := input.review.object.spec.ephemeralContainers[_]
      }
    libs:
      - |
        package lib.exempt_container

        is_exempt(container) {
            exempt_images := object.get(object.get(input, "parameters",
                {}), "exemptImages", [])
            img := container.image
            exemption := exempt_images[_]
            _matches_exemption(img, exemption)
        }

        _matches_exemption(img, exemption) {
            not endswith(exemption, "*")
            exemption == img
        }

        _matches_exemption(img, exemption) {
            endswith(exemption, "*")
            prefix := trim_suffix(exemption, "*")
            startswith(img, prefix)
        }
```

Listing A.5: Pods meet defined Pod Security Standard Constraint Template

A.6 Disable service account auto mount

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spspautomountserviceaccounttokenpod
```

```
  annotations:
    metadata.gatekeeper.sh/title: "Automount Service Account Token for
        Pod"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Controls the ability of any Pod to enable
          automountServiceAccountToken.
spec:
  crd:
    spec:
      names:
        kind: K8sPSPAutomountServiceAccountTokenPod
      validation:
        openAPIV3Schema:
          type: object
          description: >-
            Controls the ability of any Pod to enable
                automountServiceAccountToken.
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8sautomountserviceaccounttoken

        violation[{"msg": msg}] {
            obj := input.review.object
            mountServiceAccountToken(obj.spec)
            msg := sprintf("Automounting service account token is
                disallowed, pod: %v", [obj.metadata.name])
        }

        mountServiceAccountToken(spec) {
            spec.automountServiceAccountToken == true
        }

        # if there is no automountServiceAccountToken spec, check on volumeMount
             in containers. Service Account token is mounted on /var/run/secrets/
             kubernetes.io/serviceaccount
        # https://kubernetes.io/docs/reference/access-authn-authz/service-accounts
             -admin/#serviceaccount-admission-controller
        mountServiceAccountToken(spec) {
            not has_key(spec, "automountServiceAccountToken")
            "/var/run/secrets/kubernetes.io/serviceaccount" ==
                input_containers[_].volumeMounts[_].mountPath
        }

        input_containers[c] {
            c := input.review.object.spec.containers[_]
        }

        input_containers[c] {
            c := input.review.object.spec.initContainers[_]
        }

        # Ephemeral containers not checked as it is not possible to set field.

        has_key(x, k) {
            _ = x[k]
```

```
    }
```

Listing A.6: Disable automount Service Account Token

A.7 Set requests and limits for containers

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8scontainerrequests
  annotations:
    metadata.gatekeeper.sh/title: "Container Requests"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Requires containers to have memory and CPU requests set and
          constrains
      requests to be within the specified maximum values.

      https://kubernetes.io/docs/concepts/configuration/manage-resources-
          containers/
spec:
  crd:
    spec:
      names:
        kind: K8sContainerRequests
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          properties:
            exemptImages:
              description: >-
                Any container that uses an image that matches an entry in
                     this list will be excluded
                from enforcement. Prefix-matching can be signified with `*
                     `. For example: `my-image-*`.

                It is recommended that users use the fully-qualified Docker
                      image name (e.g. start with a domain name)
                in order to avoid unexpectedly exempting images from an
                     untrusted repository.
              type: array
              items:
                type: string
            cpu:
              description: "The maximum allowed cpu request on a Pod,
                  exclusive."
              type: string
            memory:
              description: "The maximum allowed memory request on a Pod,
                  exclusive."
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
```

```
package k8scontainerrequests

import data.lib.exempt_container.is_exempt

missing(obj, field) = true {
  not obj[field]
}

missing(obj, field) = true {
  obj[field] == ""
}

canonify_cpu(orig) = new {
  is_number(orig)
  new := orig * 1000
}

canonify_cpu(orig) = new {
  not is_number(orig)
  endswith(orig, "m")
  new := to_number(replace(orig, "m", ""))
}

canonify_cpu(orig) = new {
  not is_number(orig)
  not endswith(orig, "m")
  re_match("^[0-9]+(\\.[0-9]+)?$", orig)
  new := to_number(orig) * 1000
}

# 10 ** 21
mem_multiple("E") = 1000000000000000000000 { true }

# 10 ** 18
mem_multiple("P") = 1000000000000000000 { true }

# 10 ** 15
mem_multiple("T") = 1000000000000000 { true }

# 10 ** 12
mem_multiple("G") = 1000000000000 { true }

# 10 ** 9
mem_multiple("M") = 1000000000 { true }

# 10 ** 6
mem_multiple("k") = 1000000 { true }

# 10 ** 3
mem_multiple("") = 1000 { true }

# Kubernetes accepts millibyte precision when it probably shouldn't.
# https://github.com/kubernetes/kubernetes/issues/28741
# 10 ** 0
mem_multiple("m") = 1 { true }

# 1000 * 2 ** 10
```

```
mem_multiple("Ki") = 1024000 { true }

# 1000 * 2 ** 20
mem_multiple("Mi") = 1048576000 { true }

# 1000 * 2 ** 30
mem_multiple("Gi") = 1073741824000 { true }

# 1000 * 2 ** 40
mem_multiple("Ti") = 1099511627776000 { true }

# 1000 * 2 ** 50
mem_multiple("Pi") = 1125899906842624000 { true }

# 1000 * 2 ** 60
mem_multiple("Ei") = 1152921504606846976000 { true }

get_suffix(mem) = suffix {
  not is_string(mem)
  suffix := ""
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) >0
  suffix := substring(mem, count(mem) - 1, -1)
  mem_multiple(suffix)
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) >1
  suffix := substring(mem, count(mem) - 2, -1)
  mem_multiple(suffix)
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) >1
  not mem_multiple(substring(mem, count(mem) - 1, -1))
  not mem_multiple(substring(mem, count(mem) - 2, -1))
  suffix := ""
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) == 1
  not mem_multiple(substring(mem, count(mem) - 1, -1))
  suffix := ""
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) == 0
  suffix := ""
}
```

```
canonify_mem(orig) = new {
  is_number(orig)
  new := orig * 1000
}

canonify_mem(orig) = new {
  not is_number(orig)
  suffix := get_suffix(orig)
  raw := replace(orig, suffix, "")
  re_match("^[0-9]+(\\.[0-9]+)?$", raw)
  new := to_number(raw) * mem_multiple(suffix)
}

violation[{"msg": msg}] {
  general_violation[{"msg": msg, "field": "containers"}]
}

violation[{"msg": msg}] {
  general_violation[{"msg": msg, "field": "initContainers"}]
}

# Ephemeral containers not checked as it is not possible to set field.

general_violation[{"msg": msg, "field": field}] {
  container := input.review.object.spec[field][_]
  not is_exempt(container)
  cpu_orig := container.resources.requests.cpu
  not canonify_cpu(cpu_orig)
  msg := sprintf("container <%v> cpu request <%v> could not be
      parsed", [container.name, cpu_orig])
}

general_violation[{"msg": msg, "field": field}] {
  container := input.review.object.spec[field][_]
  not is_exempt(container)
  mem_orig := container.resources.requests.memory
  not canonify_mem(mem_orig)
  msg := sprintf("container <%v> memory request <%v> could not
      be parsed", [container.name, mem_orig])
}

general_violation[{"msg": msg, "field": field}] {
  container := input.review.object.spec[field][_]
  not is_exempt(container)
  not container.resources
  msg := sprintf("container <%v> has no resource requests", [
      container.name])
}

general_violation[{"msg": msg, "field": field}] {
  container := input.review.object.spec[field][_]
  not is_exempt(container)
  not container.resources.requests
  msg := sprintf("container <%v> has no resource requests", [
      container.name])
}
```

```
      general_violation[{"msg": msg, "field": field}] {
        container := input.review.object.spec[field][_]
        not is_exempt(container)
        missing(container.resources.requests, "cpu")
        msg := sprintf("container <%v> has no cpu request", [container.
            name])
      }

      general_violation[{"msg": msg, "field": field}] {
        container := input.review.object.spec[field][_]
        not is_exempt(container)
        missing(container.resources.requests, "memory")
        msg := sprintf("container <%v> has no memory request", [
            container.name])
      }

      general_violation[{"msg": msg, "field": field}] {
        container := input.review.object.spec[field][_]
        not is_exempt(container)
        cpu_orig := container.resources.requests.cpu
        cpu := canonify_cpu(cpu_orig)
        max_cpu_orig := input.parameters.cpu
        max_cpu := canonify_cpu(max_cpu_orig)
        cpu >max_cpu
        msg := sprintf("container <%v> cpu request <%v> is higher than
             the maximum allowed of <%v>", [container.name, cpu_orig,
            max_cpu_orig])
      }

      general_violation[{"msg": msg, "field": field}] {
        container := input.review.object.spec[field][_]
        not is_exempt(container)
        mem_orig := container.resources.requests.memory
        mem := canonify_mem(mem_orig)
        max_mem_orig := input.parameters.memory
        max_mem := canonify_mem(max_mem_orig)
        mem >max_mem
        msg := sprintf("container <%v> memory request <%v> is higher
            than the maximum allowed of <%v>", [container.name,
            mem_orig, max_mem_orig])
      }
    libs:
      - |
        package lib.exempt_container

        is_exempt(container) {
            exempt_images := object.get(object.get(input, "parameters",
                {}), "exemptImages", [])
            img := container.image
            exemption := exempt_images[_]
            _matches_exemption(img, exemption)
        }

        _matches_exemption(img, exemption) {
            not endswith(exemption, "*")
            exemption == img
        }
```

```
    _matches_exemption(img, exemption) {
      endswith(exemption, "*")
      prefix := trim_suffix(exemption, "*")
      startswith(img, prefix)
    }
```

Listing A.7: Requests constraint template

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8scontainerlimits
  annotations:
    metadata.gatekeeper.sh/title: "Container Limits"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Requires containers to have memory and CPU limits set and
          constrains
      limits to be within the specified maximum values.

      https://kubernetes.io/docs/concepts/configuration/manage-resources-
          containers/
spec:
  crd:
    spec:
      names:
        kind: K8sContainerLimits
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          properties:
            exemptImages:
              description: >-
                Any container that uses an image that matches an entry in
                    this list will be excluded
                from enforcement. Prefix-matching can be signified with `*
                    `. For example: `my-image-*`.

                It is recommended that users use the fully-qualified Docker
                    image name (e.g. start with a domain name)
                in order to avoid unexpectedly exempting images from an
                    untrusted repository.
              type: array
              items:
                type: string
            cpu:
              description: "The maximum allowed cpu limit on a Pod,
                  exclusive."
              type: string
            memory:
              description: "The maximum allowed memory limit on a Pod,
                  exclusive."
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
```

```
rego: |
  package k8scontainerlimits

  import data.lib.exempt_container.is_exempt

  missing(obj, field) = true {
    not obj[field]
  }

  missing(obj, field) = true {
    obj[field] == ""
  }

  canonify_cpu(orig) = new {
    is_number(orig)
    new := orig * 1000
  }

  canonify_cpu(orig) = new {
    not is_number(orig)
    endswith(orig, "m")
    new := to_number(replace(orig, "m", ""))
  }

  canonify_cpu(orig) = new {
    not is_number(orig)
    not endswith(orig, "m")
    re_match("^[0-9]+(\\.[0-9]+)?$", orig)
    new := to_number(orig) * 1000
  }

  # 10 ** 21
  mem_multiple("E") = 1000000000000000000000 { true }

  # 10 ** 18
  mem_multiple("P") = 1000000000000000000 { true }

  # 10 ** 15
  mem_multiple("T") = 1000000000000000 { true }

  # 10 ** 12
  mem_multiple("G") = 1000000000000 { true }

  # 10 ** 9
  mem_multiple("M") = 1000000000 { true }

  # 10 ** 6
  mem_multiple("k") = 1000000 { true }

  # 10 ** 3
  mem_multiple("") = 1000 { true }

  # Kubernetes accepts millibyte precision when it probably shouldn't.
  # https://github.com/kubernetes/kubernetes/issues/28741
  # 10 ** 0
  mem_multiple("m") = 1 { true }
```

```
# 1000 * 2 ** 10
mem_multiple("Ki") = 1024000 { true }

# 1000 * 2 ** 20
mem_multiple("Mi") = 1048576000 { true }

# 1000 * 2 ** 30
mem_multiple("Gi") = 1073741824000 { true }

# 1000 * 2 ** 40
mem_multiple("Ti") = 1099511627776000 { true }

# 1000 * 2 ** 50
mem_multiple("Pi") = 1125899906842624000 { true }

# 1000 * 2 ** 60
mem_multiple("Ei") = 1152921504606846976000 { true }

get_suffix(mem) = suffix {
  not is_string(mem)
  suffix := ""
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) >0
  suffix := substring(mem, count(mem) - 1, -1)
  mem_multiple(suffix)
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) >1
  suffix := substring(mem, count(mem) - 2, -1)
  mem_multiple(suffix)
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) >1
  not mem_multiple(substring(mem, count(mem) - 1, -1))
  not mem_multiple(substring(mem, count(mem) - 2, -1))
  suffix := ""
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) == 1
  not mem_multiple(substring(mem, count(mem) - 1, -1))
  suffix := ""
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) == 0
  suffix := ""
}
```

```
canonify_mem(orig) = new {
  is_number(orig)
  new := orig * 1000
}

canonify_mem(orig) = new {
  not is_number(orig)
  suffix := get_suffix(orig)
  raw := replace(orig, suffix, "")
  re_match("^[0-9]+(\\.[0-9]+)?$", raw)
  new := to_number(raw) * mem_multiple(suffix)
}

violation[{"msg": msg}] {
  general_violation[{"msg": msg, "field": "containers"}]
}

violation[{"msg": msg}] {
  general_violation[{"msg": msg, "field": "initContainers"}]
}

# Ephemeral containers not checked as it is not possible to set field.

general_violation[{"msg": msg, "field": field}] {
  container := input.review.object.spec[field][_]
  not is_exempt(container)
  cpu_orig := container.resources.limits.cpu
  not canonify_cpu(cpu_orig)
  msg := sprintf("container <%v> cpu limit <%v> could not be
      parsed", [container.name, cpu_orig])
}

general_violation[{"msg": msg, "field": field}] {
  container := input.review.object.spec[field][_]
  not is_exempt(container)
  mem_orig := container.resources.limits.memory
  not canonify_mem(mem_orig)
  msg := sprintf("container <%v> memory limit <%v> could not be
      parsed", [container.name, mem_orig])
}

general_violation[{"msg": msg, "field": field}] {
  container := input.review.object.spec[field][_]
  not is_exempt(container)
  not container.resources
  msg := sprintf("container <%v> has no resource limits", [
      container.name])
}

general_violation[{"msg": msg, "field": field}] {
  container := input.review.object.spec[field][_]
  not is_exempt(container)
  not container.resources.limits
  msg := sprintf("container <%v> has no resource limits", [
      container.name])
}
```

```
  general_violation[{"msg": msg, "field": field}] {
    container := input.review.object.spec[field][_]
    not is_exempt(container)
    missing(container.resources.limits, "cpu")
    msg := sprintf("container <%v> has no cpu limit", [container.
        name])
  }

  general_violation[{"msg": msg, "field": field}] {
    container := input.review.object.spec[field][_]
    not is_exempt(container)
    missing(container.resources.limits, "memory")
    msg := sprintf("container <%v> has no memory limit", [container
        .name])
  }

  general_violation[{"msg": msg, "field": field}] {
    container := input.review.object.spec[field][_]
    not is_exempt(container)
    cpu_orig := container.resources.limits.cpu
    cpu := canonify_cpu(cpu_orig)
    max_cpu_orig := input.parameters.cpu
    max_cpu := canonify_cpu(max_cpu_orig)
    cpu >max_cpu
    msg := sprintf("container <%v> cpu limit <%v> is higher than
        the maximum allowed of <%v>", [container.name, cpu_orig,
        max_cpu_orig])
  }

  general_violation[{"msg": msg, "field": field}] {
    container := input.review.object.spec[field][_]
    not is_exempt(container)
    mem_orig := container.resources.limits.memory
    mem := canonify_mem(mem_orig)
    max_mem_orig := input.parameters.memory
    max_mem := canonify_mem(max_mem_orig)
    mem >max_mem
    msg := sprintf("container <%v> memory limit <%v> is higher
        than the maximum allowed of <%v>", [container.name,
        mem_orig, max_mem_orig])
  }
libs:
  - |
    package lib.exempt_container

    is_exempt(container) {
        exempt_images := object.get(object.get(input, "parameters",
            {}), "exemptImages", [])
        img := container.image
        exemption := exempt_images[_]
        _matches_exemption(img, exemption)
    }

    _matches_exemption(img, exemption) {
        not endswith(exemption, "*")
        exemption == img
```

```
}

_matches_exemption(img, exemption) {
    endswith(exemption, "*")
    prefix := trim_suffix(exemption, "*")
    startswith(img, prefix)
}
```

Listing A.8: Limits constraint template

Appendix B

Cilium event logging

Detect a container running in privileged mode:


```
{
  "process_exec":{
    "process":{
      "exec_id":"bWluaWt1YmU6MTEzNzkyNjAzMjk3MjoxNzk3OA==",
      "pid":17978,
      "uid":0,
      "cwd":"/",
      "binary":"/docker-entrypoint.sh",
      "arguments":"/docker-entrypoint.sh nginx -g \"daemon off;\"",
      "flags":"execve rootcwd clone",
      "start_time":"2021-10-13T12:58:31.794Z",
      "auid":4294967295,
      "pod":{
        "namespace":"default",
        "name":"privileged-the-pod",
        "container":{
          "id":"docker://32865
              cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
              ",
          "name":"privileged-the-pod",
          "image":{
            "id":"docker-pullable://nginxsha256:644
                a70516a26004c97d0d85c7fe1d0c3a67ea8ab7ddf4aff193d9f301670cf36
                ",
            "name":"nginx:latest"
          },
          "start_time":"2021-10-13T12:58:31Z"
        }
      },
      "docker":"32865cff8fef4a9274e9fa1d",
      "parent_exec_id":"bWluaWt1YmU6MTEzNzgyOTM1MzU5NzoxNzk1OA==",
      "refcnt":1,
      "cap":{
        "permitted":[
          "CAP_CHOWN",
          "DAC_OVERRIDE",
          "CAP_DAC_READ_SEARCH",
```

```
    "CAP_FOWNER",
    "CAP_FSETID",
    "CAP_KILL",
    "CAP_SETGID",
    "CAP_SETUID",
    "CAP_SETPCAP",
    "CAP_LINUX_IMMUTABLE",
    "CAP_NET_BIND_SERVICE",
    "CAP_NET_BROADCAST",
    "CAP_NET_ADMIN",
    "CAP_NET_RAW",
    "CAP_IPC_LOCK",
    "CAP_IPC_OWNER",
    "CAP_SYS_MODULE",
    "CAP_SYS_RAWIO",
    "CAP_SYS_CHROOT",
    "CAP_SYS_PTRACE",
    "CAP_SYS_PACCT",
    "CAP_SYS_ADMIN",
    "CAP_SYS_BOOT",
    "CAP_SYS_NICE",
    "CAP_SYS_RESOURCE",
    "CAP_SYS_TIME",
    "CAP_SYS_TTY_CONFIG",
    "CAP_MKNOD",
    "CAP_LEASE",
    "CAP_AUDIT_WRITE",
    "CAP_AUDIT_CONTROL",
    "CAP_SETFCAP",
    "CAP_MAC_OVERRIDE",
    "CAP_MAC_ADMIN",
    "CAP_SYSLOG",
    "CAP_WAKE_ALARM",
    "CAP_BLOCK_SUSPEND",
    "CAP_AUDIT_READ"
  ],
  "effective":[
    "CAP_CHOWN",
    "DAC_OVERRIDE",
    "CAP_DAC_READ_SEARCH",
    "CAP_FOWNER",
    "CAP_FSETID",
    "CAP_KILL",
    "CAP_SETGID",
    "CAP_SETUID",
    "CAP_SETPCAP",
    "CAP_LINUX_IMMUTABLE",
    "CAP_NET_BIND_SERVICE",
    "CAP_NET_BROADCAST",
    "CAP_NET_ADMIN",
    "CAP_NET_RAW",
    "CAP_IPC_LOCK",
    "CAP_IPC_OWNER",
    "CAP_SYS_MODULE",
    "CAP_SYS_RAWIO",
    "CAP_SYS_CHROOT",
    "CAP_SYS_PTRACE",
```

```
      "CAP_SYS_PACCT",
      "CAP_SYS_ADMIN",
      "CAP_SYS_BOOT",
      "CAP_SYS_NICE",
      "CAP_SYS_RESOURCE",
      "CAP_SYS_TIME",
      "CAP_SYS_TTY_CONFIG",
      "CAP_MKNOD",
      "CAP_LEASE",
      "CAP_AUDIT_WRITE",
      "CAP_AUDIT_CONTROL",
      "CAP_SETFCAP",
      "CAP_MAC_OVERRIDE",
      "CAP_MAC_ADMIN",
      "CAP_SYSLOG",
      "CAP_WAKE_ALARM",
      "CAP_BLOCK_SUSPEND",
      "CAP_AUDIT_READ"
    ],
    "inheritable":[
      "CAP_CHOWN",
      "DAC_OVERRIDE",
      "CAP_DAC_READ_SEARCH",
      "CAP_FOWNER",
      "CAP_FSETID",
      "CAP_KILL",
      "CAP_SETGID",
      "CAP_SETUID",
      "CAP_SETPCAP",
      "CAP_LINUX_IMMUTABLE",
      "CAP_NET_BIND_SERVICE",
      "CAP_NET_BROADCAST",
      "CAP_NET_ADMIN",
      "CAP_NET_RAW",
      "CAP_IPC_LOCK",
      "CAP_IPC_OWNER",
      "CAP_SYS_MODULE",
      "CAP_SYS_RAWIO",
      "CAP_SYS_CHROOT",
      "CAP_SYS_PTRACE",
      "CAP_SYS_PACCT",
      "CAP_SYS_ADMIN",
      "CAP_SYS_BOOT",
      "CAP_SYS_NICE",
      "CAP_SYS_RESOURCE",
      "CAP_SYS_TIME",
      "CAP_SYS_TTY_CONFIG",
      "CAP_MKNOD",
      "CAP_LEASE",
      "CAP_AUDIT_WRITE",
      "CAP_AUDIT_CONTROL",
      "CAP_SETFCAP",
      "CAP_MAC_OVERRIDE",
      "CAP_MAC_ADMIN",
      "CAP_SYSLOG",
      "CAP_WAKE_ALARM",
      "CAP_BLOCK_SUSPEND",
```

```
        "CAP_AUDIT_READ"
      ]
    }
  },
  "parent":{
    "exec_id":"bWluaWt1YmU6MTEzNzgyOTM1MzU5NzoxNzk1OA==",
    "pid":17958,
    "uid":0,
    "cwd":"/docker/containerd/daemon/io.containerd.runtime.v1.linux/
        moby/32865
        cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
        /",
    "binary":"/usr/bin/containerd-shim",
    "arguments":"-namespace moby -workdir /var/lib/docker/containerd/
        daemon/io.containerd.runtime.v1.linux/moby/32865
        cff8fef4a9274e9fa1d80bf4 eb80d28b94e273d6d1670a6f721a9a1158
        -address /var/run/docker/containerd/containerd.sock -
        containerd-binary /usr/bin/containerd -runtime-root /var/run
        /docker/runtime-runc -systemd-cgroup",
    "flags":"execve clone",
    "start_time":"2021-10-13T12:58:31.698Z",
    "auid":4294967295,
    "parent_exec_id":"bWluaWt1YmU6NDIwODAwMDAwMDA6MjY4MA==",
    "refcnt":2
  },
  "ancestors":[
    {
      "exec_id":"bWluaWt1YmU6NDIwODAwMDAwMDA6MjY4MA==",
      "pid":2680,
      "uid":0,
      "cwd":"/",
      "binary":"/usr/bin/containerd",
      "arguments":"--config /var/run/docker/containerd/containerd.
          toml --log-level info",
      "flags":"procFS auid rootcwd",
      "start_time":"2021-10-13T12:40:15.948Z",
      "auid":0,
      "parent_exec_id":"bWluaWt1YmU6NDIwNDAwMDAwMDA6MjY3Mg==",
      "refcnt":63
    },
    {
      "exec_id":"bWluaWt1YmU6NDIwNDAwMDAwMDA6MjY3Mg==",
      "pid":2672,
      "uid":0,
      "cwd":"/",
      "binary":"/usr/bin/dockerd",
      "arguments":"-H tcp://0.0.0.0:2376 -H unix:///var/run/docker.
          sock --default-ulimit=nofile=1048576:1048576 --tlsverify
          --tlscacert /etc/docker/ca.pem --tlscert /etc/docker/
          server.pem --tlskey /etc/docker/server-key.pem --label
          provider=virtualbox --insecure-registry 10.96.0.0/12",
      "flags":"procFS auid rootcwd",
      "start_time":"2021-10-13T12:40:15.908Z",
      "auid":0,
      "parent_exec_id":"bWluaWt1YmU6MjEwMDAwMDAwOjE=",
      "refcnt":65
    },
```

```
      {
        "exec_id":"bWluaWt1YmU6MjEwMDAwMDAwOjE=",
        "pid":1,
        "uid":0,
        "cwd":"/",
        "binary":"/usr/lib/systemd/systemd",
        "arguments":"noembed norestore",
        "flags":"procFS auid rootcwd",
        "start_time":"2021-10-13T12:39:34.078Z",
        "auid":0,
        "refcnt":101
      }
    ]
  },
  "node_name":"minikube",
  "time":"2021-10-13T12:58:31.794Z"
}
```


Listing B.1: Detect privileged container

As a second step, the attacker can use kubectl exec to get shell access to `privileged-the-pod`:


```
{
  "process_exec":{
    "process":{
      "exec_id":"bWluaWt1YmU6MTI5NDM3OTU0NzQ3ODoxOTU5NA==",
      "pid":19594,
      "uid":0,
      "cwd":"/",
      "binary":"/bin/bash",
      "flags":"execve rootcwd clone",
      "start_time":"2021-10-13T13:01:08.248Z",
      "auid":4294967295,
      "pod":{
        "namespace":"default",
        "name":"privileged-the-pod",
        "container":{
          "id":"docker://32865
              cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
              ",
          "name":"privileged-the-pod",
          "image":{
            "id":"docker-pullable://nginxsha256:644
                a70516a26004c97d0d85c7fe1d0c3a67ea8ab7ddf4aff193d9f301670cf36
                ",
            "name":"nginx:latest"
          },
          "start_time":"2021-10-13T12:58:31Z"
        }
      },
      "docker":"32865cff8fef4a9274e9fa1d",
      "parent_exec_id":"bWluaWt1YmU6MTI5NDMzMzczMTY4NToxOTU4NA==",
      "refcnt":1,
      "cap":{
        "permitted":[
          "CAP_CHOWN",
          "DAC_OVERRIDE",
```

```
    "CAP_DAC_READ_SEARCH",
    "CAP_FOWNER",
    "CAP_FSETID",
    "CAP_KILL",
    "CAP_SETGID",
    "CAP_SETUID",
    "CAP_SETPCAP",
    "CAP_LINUX_IMMUTABLE",
    "CAP_NET_BIND_SERVICE",
    "CAP_NET_BROADCAST",
    "CAP_NET_ADMIN",
    "CAP_NET_RAW",
    "CAP_IPC_LOCK",
    "CAP_IPC_OWNER",
    "CAP_SYS_MODULE",
    "CAP_SYS_RAWIO",
    "CAP_SYS_CHROOT",
    "CAP_SYS_PTRACE",
    "CAP_SYS_PACCT",
    "CAP_SYS_ADMIN",
    "CAP_SYS_BOOT",
    "CAP_SYS_NICE",
    "CAP_SYS_RESOURCE",
    "CAP_SYS_TIME",
    "CAP_SYS_TTY_CONFIG",
    "CAP_MKNOD",
    "CAP_LEASE",
    "CAP_AUDIT_WRITE",
    "CAP_AUDIT_CONTROL",
    "CAP_SETFCAP",
    "CAP_MAC_OVERRIDE",
    "CAP_MAC_ADMIN",
    "CAP_SYSLOG",
    "CAP_WAKE_ALARM",
    "CAP_BLOCK_SUSPEND",
    "CAP_AUDIT_READ"
  ],
  "effective":[
    "CAP_CHOWN",
    "DAC_OVERRIDE",
    "CAP_DAC_READ_SEARCH",
    "CAP_FOWNER",
    "CAP_FSETID",
    "CAP_KILL",
    "CAP_SETGID",
    "CAP_SETUID",
    "CAP_SETPCAP",
    "CAP_LINUX_IMMUTABLE",
    "CAP_NET_BIND_SERVICE",
    "CAP_NET_BROADCAST",
    "CAP_NET_ADMIN",
    "CAP_NET_RAW",
    "CAP_IPC_LOCK",
    "CAP_IPC_OWNER",
    "CAP_SYS_MODULE",
    "CAP_SYS_RAWIO",
    "CAP_SYS_CHROOT",
```

```
    "CAP_SYS_PTRACE",
    "CAP_SYS_PACCT",
    "CAP_SYS_ADMIN",
    "CAP_SYS_BOOT",
    "CAP_SYS_NICE",
    "CAP_SYS_RESOURCE",
    "CAP_SYS_TIME",
    "CAP_SYS_TTY_CONFIG",
    "CAP_MKNOD",
    "CAP_LEASE",
    "CAP_AUDIT_WRITE",
    "CAP_AUDIT_CONTROL",
    "CAP_SETFCAP",
    "CAP_MAC_OVERRIDE",
    "CAP_MAC_ADMIN",
    "CAP_SYSLOG",
    "CAP_WAKE_ALARM",
    "CAP_BLOCK_SUSPEND",
    "CAP_AUDIT_READ"
  ],
  "inheritable":[
    "CAP_CHOWN",
    "DAC_OVERRIDE",
    "CAP_DAC_READ_SEARCH",
    "CAP_FOWNER",
    "CAP_FSETID",
    "CAP_KILL",
    "CAP_SETGID",
    "CAP_SETUID",
    "CAP_SETPCAP",
    "CAP_LINUX_IMMUTABLE",
    "CAP_NET_BIND_SERVICE",
    "CAP_NET_BROADCAST",
    "CAP_NET_ADMIN",
    "CAP_NET_RAW",
    "CAP_IPC_LOCK",
    "CAP_IPC_OWNER",
    "CAP_SYS_MODULE",
    "CAP_SYS_RAWIO",
    "CAP_SYS_CHROOT",
    "CAP_SYS_PTRACE",
    "CAP_SYS_PACCT",
    "CAP_SYS_ADMIN",
    "CAP_SYS_BOOT",
    "CAP_SYS_NICE",
    "CAP_SYS_RESOURCE",
    "CAP_SYS_TIME",
    "CAP_SYS_TTY_CONFIG",
    "CAP_MKNOD",
    "CAP_LEASE",
    "CAP_AUDIT_WRITE",
    "CAP_AUDIT_CONTROL",
    "CAP_SETFCAP",
    "CAP_MAC_OVERRIDE",
    "CAP_MAC_ADMIN",
    "CAP_SYSLOG",
    "CAP_WAKE_ALARM",
```

```
        "CAP_BLOCK_SUSPEND",
        "CAP_AUDIT_READ"
      ]
    }
  },
  "parent":{
    "exec_id":"bWluaWt1YmU6MTI5NDMzMzczMTY4NToxOTU4NA==",
    "pid":19584,
    "uid":0,
    "cwd":"/docker/containerd/daemon/io.containerd.runtime.v1.linux/
        moby/32865
        cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
        /",
    "binary":"/usr/bin/runc",
    "arguments":"--root /var/run/docker/runtime-runc/moby --log /run/
        docker/containerd/daemon/io.containerd.runtime.v1.linux/moby
        /32865cff8fef4a9274e9fa1d80bf48eb8
        d28b94e273d6d1670a6f721a9a1158/log.json --log-format json --
        systemd-cgroup exec --process /tmp/runc-process133903661 --
        console-socket /tmp/pty028492678/pty.sock --detach --pid-
        file /run/docker/containerd/daemon/io.containerd.runtime.v1.
        linux/moby/32865cff8fef4a9274e9fa1d80bf48eb8
        d28b94e273d6d1670a6f721a9a1158/
        a5579f11fb7d75d66213f488bf44e9c37b92c196dee5e94647e6f60c59cf6693
        .pid 32865
        cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
        ",
    "flags":"execve clone",
    "start_time":"2021-10-13T13:01:08.202Z",
    "auid":4294967295,
    "parent_exec_id":"bWluaWt1YmU6MTEzNzgyOTM1MzU5NzoxNzk1OA==",
    "refcnt":2
  },
  "ancestors":[
    {
      "exec_id":"bWluaWt1YmU6MTEzNzgyOTM1MzU5NzoxNzk1OA==",
      "pid":17958,
      "uid":0,
      "cwd":"/docker/containerd/daemon/io.containerd.runtime.v1.
          linux/moby/32865
          cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
          /",
      "binary":"/usr/bin/containerd-shim",
      "arguments":"-namespace moby -workdir /var/lib/docker/
          containerd/daemon/io.containerd.runtime.v1.linux/moby
          /32865cff8fef4a9274e9fa1d80bf4
          eb80d28b94e273d6d1670a6f721a9a1158 -address /var/run/
          docker/containerd/containerd.sock -containerd-binary /usr/
          bin/containerd -runtime-root /var/run/docker/runtime-runc
          -systemd-cgroup",
      "flags":"execve clone",
      "start_time":"2021-10-13T12:58:31.698Z",
      "auid":4294967295,
      "parent_exec_id":"bWluaWt1YmU6NDIwODAwMDAwMDA6MjY4MA==",
      "refcnt":4
    },
    {
```

```
        "exec_id":"bWluaWt1YmU6NDIwODAwMDAwMDA6MjY4MA==",
        "pid":2680,
        "uid":0,
        "cwd":"/",
        "binary":"/usr/bin/containerd",
        "arguments":"--config /var/run/docker/containerd/containerd.
            toml --log-level info",
        "flags":"procFS auid rootcwd",
        "start_time":"2021-10-13T12:40:15.948Z",
        "auid":0,
        "parent_exec_id":"bWluaWt1YmU6NDIwNDAwMDAwMDA6MjY3Mg==",
        "refcnt":65
      },
      {
        "exec_id":"bWluaWt1YmU6NDIwNDAwMDAwMDA6MjY3Mg==",
        "pid":2672,
        "uid":0,
        "cwd":"/",
        "binary":"/usr/bin/dockerd",
        "arguments":"-H tcp://0.0.0.0:2376 -H unix:///var/run/docker.
            sock --default-ulimit=nofile=1048576:1048576 --tlsverify
            --tlscacert /etc/docker/ca.pem --tlscert /etc/docker/
            server.pem --tlskey /etc/docker/server-key.pem --label
            provider=virtualbox --insecure-registry 10.96.0.0/12",
        "flags":"procFS auid rootcwd",
        "start_time":"2021-10-13T12:40:15.908Z",
        "auid":0,
        "parent_exec_id":"bWluaWt1YmU6MjEwMDAwMDAwOjE=",
        "refcnt":67
      },
      {
        "exec_id":"bWluaWt1YmU6MjEwMDAwMDAwOjE=",
        "pid":1,
        "uid":0,
        "cwd":"/",
        "binary":"/usr/lib/systemd/systemd",
        "arguments":"noembed norestore",
        "flags":"procFS auid rootcwd",
        "start_time":"2021-10-13T12:39:34.078Z",
        "auid":0,
        "refcnt":106
      }
    ]
  },
  "node_name":"minikube",
  "time":"2021-10-13T13:01:08.248Z"
}
```

Listing B.2: Container executing a shell

Finally the attacker enters the host via `nsenter` command:

```
{
  "process_exec":{
    "process":{
      "exec_id":"bWluaWt1YmU6MTYyNzIwNjE3NjIzMjoyMzc0Nw==",
      "pid":23747,
      "uid":0,
```

```
"cwd":"/",
"binary":"/usr/bin/bash",
"flags":"execve rootcwd clone",
"start_time":"2021-10-13T13:06:41.074Z",
"auid":4294967295,
"pod":{
  "namespace":"default",
  "name":"privileged-the-pod",
  "container":{
    "id":"docker://32865
        cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
        ",
    "name":"privileged-the-pod",
    "image":{
      "id":"docker-pullable://nginxsha256:644
          a70516a26004c97d0d85c7fe1d0c3a67ea8ab7ddf4aff193d9f301670cf36
          ",
      "name":"nginx:latest"
    },
    "start_time":"2021-10-13T12:58:31Z"
  }
},
"docker":"32865cff8fef4a9274e9fa1d",
"parent_exec_id":"bWluaWt1YmU6MTYyNzIwMjkzMjkyMToyMzc0Ng==",
"refcnt":1,
"cap":{
  "permitted":[
    "CAP_CHOWN",
    "DAC_OVERRIDE",
    "CAP_DAC_READ_SEARCH",
    "CAP_FOWNER",
    "CAP_FSETID",
    "CAP_KILL",
    "CAP_SETGID",
    "CAP_SETUID",
    "CAP_SETPCAP",
    "CAP_LINUX_IMMUTABLE",
    "CAP_NET_BIND_SERVICE",
    "CAP_NET_BROADCAST",
    "CAP_NET_ADMIN",
    "CAP_NET_RAW",
    "CAP_IPC_LOCK",
    "CAP_IPC_OWNER",
    "CAP_SYS_MODULE",
    "CAP_SYS_RAWIO",
    "CAP_SYS_CHROOT",
    "CAP_SYS_PTRACE",
    "CAP_SYS_PACCT",
    "CAP_SYS_ADMIN",
    "CAP_SYS_BOOT",
    "CAP_SYS_NICE",
    "CAP_SYS_RESOURCE",
    "CAP_SYS_TIME",
    "CAP_SYS_TTY_CONFIG",
    "CAP_MKNOD",
    "CAP_LEASE",
    "CAP_AUDIT_WRITE",
```

```
    "CAP_AUDIT_CONTROL",
    "CAP_SETFCAP",
    "CAP_MAC_OVERRIDE",
    "CAP_MAC_ADMIN",
    "CAP_SYSLOG",
    "CAP_WAKE_ALARM",
    "CAP_BLOCK_SUSPEND",
    "CAP_AUDIT_READ"
  ],
  "effective":[
    "CAP_CHOWN",
    "DAC_OVERRIDE",
    "CAP_DAC_READ_SEARCH",
    "CAP_FOWNER",
    "CAP_FSETID",
    "CAP_KILL",
    "CAP_SETGID",
    "CAP_SETUID",
    "CAP_SETPCAP",
    "CAP_LINUX_IMMUTABLE",
    "CAP_NET_BIND_SERVICE",
    "CAP_NET_BROADCAST",
    "CAP_NET_ADMIN",
    "CAP_NET_RAW",
    "CAP_IPC_LOCK",
    "CAP_IPC_OWNER",
    "CAP_SYS_MODULE",
    "CAP_SYS_RAWIO",
    "CAP_SYS_CHROOT",
    "CAP_SYS_PTRACE",
    "CAP_SYS_PACCT",
    "CAP_SYS_ADMIN",
    "CAP_SYS_BOOT",
    "CAP_SYS_NICE",
    "CAP_SYS_RESOURCE",
    "CAP_SYS_TIME",
    "CAP_SYS_TTY_CONFIG",
    "CAP_MKNOD",
    "CAP_LEASE",
    "CAP_AUDIT_WRITE",
    "CAP_AUDIT_CONTROL",
    "CAP_SETFCAP",
    "CAP_MAC_OVERRIDE",
    "CAP_MAC_ADMIN",
    "CAP_SYSLOG",
    "CAP_WAKE_ALARM",
    "CAP_BLOCK_SUSPEND",
    "CAP_AUDIT_READ"
  ],
  "inheritable":[
    "CAP_CHOWN",
    "DAC_OVERRIDE",
    "CAP_DAC_READ_SEARCH",
    "CAP_FOWNER",
    "CAP_FSETID",
    "CAP_KILL",
    "CAP_SETGID",
```

```
          "CAP_SETUID",
          "CAP_SETPCAP",
          "CAP_LINUX_IMMUTABLE",
          "CAP_NET_BIND_SERVICE",
          "CAP_NET_BROADCAST",
          "CAP_NET_ADMIN",
          "CAP_NET_RAW",
          "CAP_IPC_LOCK",
          "CAP_IPC_OWNER",
          "CAP_SYS_MODULE",
          "CAP_SYS_RAWIO",
          "CAP_SYS_CHROOT",
          "CAP_SYS_PTRACE",
          "CAP_SYS_PACCT",
          "CAP_SYS_ADMIN",
          "CAP_SYS_BOOT",
          "CAP_SYS_NICE",
          "CAP_SYS_RESOURCE",
          "CAP_SYS_TIME",
          "CAP_SYS_TTY_CONFIG",
          "CAP_MKNOD",
          "CAP_LEASE",
          "CAP_AUDIT_WRITE",
          "CAP_AUDIT_CONTROL",
          "CAP_SETFCAP",
          "CAP_MAC_OVERRIDE",
          "CAP_MAC_ADMIN",
          "CAP_SYSLOG",
          "CAP_WAKE_ALARM",
          "CAP_BLOCK_SUSPEND",
          "CAP_AUDIT_READ"
        ]
      }
    },
    "parent":{
      "exec_id":"bWluaWt1YmU6MTYyNzIwMjkzMjkyMTToyMzc0Ng==",
      "pid":23746,
      "uid":0,
      "cwd":"/",
      "binary":"/usr/bin/nsenter",
      "arguments":"-t 1 -a bash",
      "flags":"execve rootcwd clone",
      "start_time":"2021-10-13T13:06:41.071Z",
      "auid":4294967295,
      "pod":{
        "namespace":"default",
        "name":"privileged-the-pod",
        "container":{
          "id":"docker://32865
              cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
              ",
          "name":"privileged-the-pod",
          "image":{
            "id":"docker-pullable://nginxsha256:644
                a70516a26004c97d0d85c7fe1d0c3a67ea8ab7ddf4aff193d9f301670cf36
                ",
            "name":"nginx:latest"
```

```
        },
        "start_time":"2021-10-13T12:58:31Z"
      }
    },
    "docker":"32865cff8fef4a9274e9fa1d",
    "parent_exec_id":"bWluaWt1YmU6MTI5NDM3OTU0NzQ3ODoxOTU5NA==",
    "refcnt":2
  },
  "ancestors":[
    {
      "exec_id":"bWluaWt1YmU6MTI5NDM3OTU0NzQ3ODoxOTU5NA==",
      "pid":19594,
      "uid":0,
      "cwd":"/",
      "binary":"/bin/bash",
      "flags":"execve rootcwd clone",
      "start_time":"2021-10-13T13:01:08.248Z",
      "auid":4294967295,
      "pod":{
        "namespace":"default",
        "name":"privileged-the-pod",
        "container":{
          "id":"docker://32865
              cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
              ",
          "name":"privileged-the-pod",
          "image":{
            "id":"docker-pullable://nginxsha256:644
                a70516a26004c97d0d85c7fe1d0c3a67ea8ab7ddf4aff193d9f301670cf36
                ",
            "name":"nginx:latest"
          },
          "start_time":"2021-10-13T12:58:31Z"
        }
      },
      "docker":"32865cff8fef4a9274e9fa1d",
      "parent_exec_id":"bWluaWt1YmU6MTI5NDMzMzczMTY4NToxOTU4NA==",
      "refcnt":3
    },
    {
      "exec_id":"bWluaWt1YmU6MTI5NDMzMzczMTY4NToxOTU4NA==",
      "pid":19584,
      "uid":0,
      "cwd":"/docker/containerd/daemon/io.containerd.runtime.v1.
          linux/moby/32865
          cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
          /",
      "binary":"/usr/bin/runc",
      "arguments":"--root /var/run/docker/runtime-runc/moby --log /
          run/docker/containerd/daemon/io.containerd.runtime.v1.
          linux/moby/32865cff8fef4a9274e9fa1d80bf48eb8
          d28b94e273d6d1670a6f721a9a1158/log.json --log-format json
          --systemd-cgroup exec --process /tmp/runc-process133903661
           --console-socket /tmp/pty028492678/pty.sock --detach --
          pid-file /run/docker/containerd/daemon/io.containerd.
          runtime.v1.linux/moby/32865cff8fef4a9274e9fa1d80bf48eb8
          d28b94e273d6d1670a6f721a9a1158/
```

```
        a5579f11fb7d75d66213f488bf44e9c37b92c196dee5e94647e6f60c59cf6693
        .pid 32865
        cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
        ",
    "flags":"execve clone",
    "start_time":"2021-10-13T13:01:08.202Z",
    "auid":4294967295,
    "parent_exec_id":"bWluaWt1YmU6MTEzNzgyOTM1MzU5NzoxNzk1OA==",
    "refcnt":3
  },
  {
    "exec_id":"bWluaWt1YmU6MTEzNzgyOTM1MzU5NzoxNzk1OA==",
    "pid":17958,
    "uid":0,
    "cwd":"/docker/containerd/daemon/io.containerd.runtime.v1.
        linux/moby/32865
        cff8fef4a9274e9fa1d80bf48eb80d28b94e273d6d1670a6f721a9a1158
        /",
    "binary":"/usr/bin/containerd-shim",
    "arguments":"-namespace moby -workdir /var/lib/docker/
        containerd/daemon/io.containerd.runtime.v1.linux/moby
        /32865cff8fef4a9274e9fa1d80bf4
        eb80d28b94e273d6d1670a6f721a9a1158 -address /var/run/
        docker/containerd/containerd.sock -containerd-binary /usr/
        bin/containerd -runtime-root /var/run/docker/runtime-runc
        -systemd-cgroup",
    "flags":"execve clone",
    "start_time":"2021-10-13T12:58:31.698Z",
    "auid":4294967295,
    "parent_exec_id":"bWluaWt1YmU6NDIwODAwMDAwMDA6MjY4MA==",
    "refcnt":5
  },
  {
    "exec_id":"bWluaWt1YmU6NDIwODAwMDAwMDA6MjY4MA==",
    "pid":2680,
    "uid":0,
    "cwd":"/",
    "binary":"/usr/bin/containerd",
    "arguments":"--config /var/run/docker/containerd/containerd.
        toml --log-level info",
    "flags":"procFS auid rootcwd",
    "start_time":"2021-10-13T12:40:15.948Z",
    "auid":0,
    "parent_exec_id":"bWluaWt1YmU6NDIwNDAwMDAwMDA6MjY3Mg==",
    "refcnt":87
  },
  {
    "exec_id":"bWluaWt1YmU6NDIwNDAwMDAwMDA6MjY3Mg==",
    "pid":2672,
    "uid":0,
    "cwd":"/",
    "binary":"/usr/bin/dockerd",
    "arguments":"-H tcp://0.0.0.0:2376 -H unix:///var/run/docker.
        sock --default-ulimit=nofile=1048576:1048576 --tlsverify
        --tlscacert /etc/docker/ca.pem --tlscert /etc/docker/
        server.pem --tlskey /etc/docker/server-key.pem --label
        provider=virtualbox --insecure-registry 10.96.0.0/12",
```

```
        "flags":"procFS auid rootcwd",
        "start_time":"2021-10-13T12:40:15.908Z",
        "auid":0,
        "parent_exec_id":"bWluaWt1YmU6MjEwMDAwMDAwOjE=",
        "refcnt":89
      },
      {
        "exec_id":"bWluaWt1YmU6MjEwMDAwMDAwOjE=",
        "pid":1,
        "uid":0,
        "cwd":"/",
        "binary":"/usr/lib/systemd/systemd",
        "arguments":"noembed norestore",
        "flags":"procFS auid rootcwd",
        "start_time":"2021-10-13T12:39:34.078Z",
        "auid":0,
        "refcnt":128
      }
    ]
  },
  "node_name":"minikube",
  "time":"2021-10-13T13:06:41.074Z"
}
```


Listing B.3: Container entering host via nsenter command

Appendix C

Tetragon Tracing Policies examples

The `DnsLookup` action can be used to perform a remote interaction such as triggering Thinkst canaries or any system that can be triggered via an DNS entry request. It uses the `argFqdn` field to specify the domain to lookup.

```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: "dns"
spec:
  kprobes:
  - call: "fd_install"
    syscall: false
    args:
    - index: 0
      type: int
    - index: 1
      type: "file"
    selectors:
    - matchArgs:
      - index: 1
        operator: "Equal"
        values:
        - "/etc/passwd"
      matchActions:
      - action: DnsLookup
        argFqdn: ebpf.io
```

Listing C.1: Trigger Canary via DnsLookup action

`Override` action allows to modify the return value of call. While `Sigkill` will terminate the entire process responsible for making the call, `Override` will override the return value that was supposed to be returned with the value given in the argError field. It's then up to the process handling of the return value of the function to stop or continue the execution.

```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: "sys-linkat-passwd"
spec:
  kprobes:
  - call: "sys_linkat"
    syscall: true
    args:
    - index: 0
      type: "int"
    - index: 1
      type: "string"
    - index: 2
      type: "int"
    - index: 3
      type: "string"
    - index: 4
      type: "int"
    selectors:
    - matchArgs:
      - index: 1
        operator: "Equal"
        values:
        - "/etc/passwd\0"
      matchActions:
      - action: Override
        argError: -1
```

Listing C.2: Override return value of command via Override action

Trace `tcp_connect`, `tcp_close` and `tcp_sendmsg` for CIDR blocks 127.0.0.1/8 and 192.168.0.0/16.

```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: "connect"
spec:
  kprobes:
  - call: "tcp_connect"
    syscall: false
    args:
    - index: 0
      type: "sock"
    selectors:
    - matchArgs:
      - index: 0
        operator: "DAddr"
        values:
        - "127.0.0.1/8"
        - "192.168.0.0/16"
  - call: "tcp_close"
    syscall: false
    args:
    - index: 0
      type: "sock"
```

```
  selectors:
  - matchArgs:
    - index: 0
      operator: "DAddr"
      values:
      - "127.0.0.1/8"
      - "192.168.0.0/16"
- call: "tcp_sendmsg"
  syscall: false
  args:
  - index: 0
    type: "sock"
  - index: 2
    type: int
  selectors:
  - matchArgs:
    - index: 0
      operator: "DAddr"
      values:
      - "127.0.0.1/8"
      - "192.168.0.0/16’’
```

Listing C.3: Trace TCP calls for specific CIDR blocks