



# Entwurf eines Einsatzkonzeptes für Monitoring- und Loggingsysteme zur Angriffsdetektion in Kubernetes- orchestrierten Container - Infrastrukturen



# Danksagung

Diese Studie wurde von Herrn Sebastian Kawelke als Abschlussarbeit für den Bachelor of Science an der Hochschule Bonn-Rhein-Sieg verfasst. Betreut wurde er dabei vom Referat TK 22 – Virtualisierung und Cloud-Sicherheit des BSI, Frau Prof. Dr. Kerstin Uhde (H-BRS, Prüferin) und Herrn Prof. Dr. Martin Leischner (H-BRS, Prüfer).

Das Thema der Studie wurde nach einem längeren Praktikum des Autors in dem Referat TK 22 und der intensiven Mitarbeit an Beratungstätigkeiten verschiedener Institutionen ausgesucht. Dort wurde festgestellt, dass die Säule Detektion beim Einsatz von Kubernetes häufig unzureichend betrachtet wird.

Die vorliegende Studie behandelt daher das Thema Kubernetes-Monitoring und beschreibt beispielhafte Lösungsansätze. Diese sind als Orientierung und Hilfestellung zu verstehen; die Studie erhebt daher keinen Anspruch auf Vollständigkeit oder Allgemeingültigkeit. Das Dokument ersetzt im Rahmen eines konkreten Einsatzes von Kubernetes nicht die Erstellung eines angepassten Detailkonzeptes.

Das BSI dankt Herrn Sebastian Kawelke für das Erstellen dieser Studie.



# Kurzzusammenfassung

Diese Arbeit betrachtet die Möglichkeiten und konkreten Maßnahmen zur Detektion von für eine Kubernetes-orchestrierte Container-Infrastruktur spezifischen Bedrohungen. Ziel der Arbeit ist es, dass Entwerfen einer Logginginfrastruktur und eines Loggingkonzepts für die Detektion von Angriffen, durch eine strukturierte Modellierung von Angriffsszenarien und durch Aufzeigen von generellen, aber auch sehr konkreten Maßnahmen, allgemein zu erleichtern.

Um dieses Ziel zu erreichen, werden auf Basis einer Bedrohungsmatrix für Kubernetes (entwickelt von Microsoft) relevante Angriffe identifiziert und analysiert. Es wird betrachtet, welche Gefahr von den ausgewählten Bedrohungen ausgeht und wie diese durch Loggingmaßnahmen zu erkennen sind. Dazu wird bestimmt, welche Ansätze zur Erkennung möglich sind und diese dann durch konkrete Maßnahmen angereichert. Es ergibt sich so ein umfassender Überblick über zu berücksichtigende Angriffstypen und eine Hilfestellung bei dem Aufbau eines Loggingsystems zur Erkennung von Bedrohungen

Die theoretischen Aussagen werden durch die praktische Umsetzung ausgewählter repräsentativer Angriffe auf ihre Wirksamkeit geprüft und evaluiert.



# Inhalt

1	Einleitung.....	9
1.1	Motivation.....	10
1.2	Vision.....	10
1.3	Ziel der Arbeit .....	11
2	Grundlagen.....	12
2.1	Containerisierung & Kubernetes.....	12
2.1.1	Containervirtualisierung.....	13
2.1.2	Kubernetes .....	13
2.2	Logging & Monitoring .....	17
2.2.1	Definitionen .....	17
2.2.2	Dezentrales und zentralisiertes Logging.....	18
2.2.3	Sicherheit bei der Übertragung von Log-Daten .....	20
2.2.4	Elastic-Stack - Zentralisierte Logging-Architektur .....	21
2.2.5	Falco - Kubernetes threat detection engine .....	23
3	Konzept .....	24
3.1	Identifizierung relevanter Störungen und Angriffe .....	24
3.2	Operationalisierung, Identifizierung von Indikatoren und Empfehlung von Maßnahmen .....	27
3.2.1	(IZ) Initialer Zugriff.....	27
3.2.2	(AU) Ausführung.....	32
3.2.3	(PS) Persistence.....	35
3.2.4	(PE) Privilegien-Eskalation.....	38
3.2.5	(UV) Umgehung der Verteidigung .....	40
3.2.6	(ZA) Zugang zu Anmelde-informationen.....	42
3.2.7	(EK) Erkundung.....	45
3.2.8	(LM) Lateral Movement .....	47
4	Praktische Umsetzung.....	50
4.1	Infrastruktur & Aufbau des Experiments.....	50
4.1.1	Elastic-Stack basiertes zentrales Logging.....	51
4.1.2	Falco und Zusatz-Dienste .....	51
4.1.3	Infrastruktur-Übersicht.....	52
4.2	Durchführung.....	52
4.2.1	bash/cmd innerhalb des Containers (AU.2).....	52
4.2.2	Neuer Container (AU.3).....	57
4.2.3	Schreibbarer Host-Pfad-Mount (PS.2).....	59
4.2.4	Kubernetes CronJob (PS.3).....	63
4.2.5	Container-Logs löschen (UV.1).....	64

4.2.6	Netzwerk-Mapping bzw. Cluster-internes Networking (EK.3 bzw. LM.2).....	67
5	Evaluierung.....	70
6	Zusammenfassung und Fazit .....	72
7	Literaturverzeichnis.....	73
8	Abbildungsverzeichnis.....	77





# 1 Einleitung

## 1.1 Motivation

Aufgrund der steigenden Komplexität von Softwaresystemen entwickelten sich neue Architekturmuster ([SMMR16]). Vor allem die Microservice-Architektur ist inzwischen einer der vorherrschenden Ansätze ([IS18]). Wesentliche Eigenschaft dieser Architektur ist die Aufspaltung von (ehemals monolithischen) Systemen auf Basis der Geschäftslogik. Angestrebt wird der Aufbau eines Softwaresystems als eine Sammlung unabhängiger, autonomer Dienste, die lose gekoppelt sind. Die einzelnen Dienste sollen unabhängig voneinander entwickelt, bereitgestellt und skaliert werden können ([IS18]). Technisch realisiert wird dies in der Entwicklung und im Betrieb vorwiegend durch den Einsatz der Containervirtualisierung.

1979 wurde mit 'chroot' in UNIX Version 7 der Grundstein für die containerisierte Isolierung von Anwendungen geschaffen. Der Ansatz der Containervirtualisierung wurde in verschiedenen Formen weiterentwickelt und erlangte zunehmend, vorrangig bei großen Unternehmen, Relevanz. Eine breite Bekanntheit erreichte die Containervirtualisierung allerdings erst mit der Veröffentlichung von Docker im Jahr 2013. Docker basiert auf den 'Linux Containern' (LXC), welche durch 'Linux cgroups' und 'namespaces' eingeführt wurden ([IS18]).

Der Microservice-Ansatz, welcher zeitlich nach den Grundlagen der Containervirtualisierung entwickelt wurde, hat sich im praktischen Einsatz der flexiblen Trennung auf Basis von Containern bedient. In der Entwicklung, sowie im Betrieb werden die Dienste in unabhängigen Containern betrieben. Um diese Vielzahl an Containern gut verwalten zu können und Anforderungen an Skalierung und Verfügbarkeit umzusetzen, werden Orchestrierungssysteme wie Kubernetes eingesetzt ([VSTK19]).

Diese infrastrukturellen Änderungen haben auch Implikationen für die Informationssicherheit. Als Vorteil ergibt sich, dass die Spezialisierung der einzelnen Microservices und ihrer Container es erleichtert, Störungen und Angriffe zu identifizieren. Die erhöhte Komplexität der Systeme (insbesondere durch hinzugekommene Managementebenen) hingegen vergrößert auch die Angriffsfläche.

## 1.2 Vision

Bei einer Infrastruktur auf Basis von Containervirtualisierung und dem Microservicedesign kann jeder Dienst und sein Container sowie die übergreifenden Orchestrierungs-Systeme, Netzwerkkomponenten und Host-Systeme jeweils Informationen über die aktuellen Betriebsparameter und Ereignisse liefern. Dadurch ergibt sich eine Vielzahl an einzelnen Informationsströmen aus verschiedenen Blickwinkeln und hinsichtlich verschiedener Ebenen. In der Praxis ist es ein Vorteil diese einzelnen Ströme in einem zentralisierten Logging- und Monitoringsystem zusammenzuführen. Dies erfolgt durch den Einsatz von bewährten Systemen (z.B. Elastic Stack, [Ela21c]). Die Motivation hinter dieser Informationssammlung ist je nach Stakeholder unterschiedlich. Entwicklerteams wollen insbesondere Informationen bezüglich des Betriebes ihrer Anwendung sammeln, um diese zu optimieren. Administrationsteams wollen den Betrieb grundsätzlich überwachen und auf Ausfälle reagieren. Ein wesentlicher Einsatzzweck von Logging ist auch die IT-Sicherheit, genauer gesagt die Detektion im Rahmen einer Sammlung von weiteren IT-Sicherheitsmaßnahmen. Hier liegt das Bestreben darin, anhand von Logging- und Monitoring-Daten Angriffe zu erkennen und im Nachgang analysieren zu können.

Da es aufgrund der Komplexität moderner Systeme schwierig ist, eine für die IT-Sicherheit passende Logging-Struktur zu entwickeln, soll diese Arbeit in Bezug auf die häufig verwendete

Orchestrierungs-Plattform Kubernetes eine Hilfestellung bieten. Da die Ausgestaltung durch die Verwendung von Containern und einer geeigneten Orchestrierung technisch standardisiert ist, ist es möglich, applikationsagnostisch Maßnahmen zu definieren. Korrekt eingesetzt, bietet dies die Chance, Detektionsmöglichkeiten, die durch Logging realisiert sind, deutlich zu verbessern und einer breiteren Masse zugänglich zu machen. Hierfür wurden neben den Ausarbeitungen und praktischen Prüfungen in dieser Arbeit, existierende Klassifikationen und Analysen herangezogen (siehe Kapitel 3, Microsoft Bedrohungsmatrix für Kubernetes, Abbildung 3.1).

Protokollierungsdaten sind nicht nur bei der Detektion von hoher Relevanz, sondern auch bei der Aufarbeitung von Vorfällen und einer anschließenden Verbesserung der Präventionsmaßnahmen. Daher müssen die Ansprüche an eine sichere und integre Übertragung der Daten im Sinne der digitalen Forensik ebenfalls berücksichtigt werden.

### 1.3 Ziel der Arbeit

Ziel der Arbeit ist es, anwendungsunabhängige, aber Container- und Kubernetes- spezifische Abweichungen zu identifizieren, welche auf relevante Störungen oder Angriffe (IoC, Indicators of Compromise) hindeuten und für diese ein allgemeingültiges Konzept zur Erkennung mittels Logging-Systemen zu entwickeln. Es wird dafür die von Microsoft entwickelte Bedrohungsmatrix für Kubernetes ([MW20], basierend auf dem MITRE ATT&CK Rahmenwerk) herangezogen. Für die Indikatoren soll dargestellt werden, wie sie mithilfe etablierter Logging- und Monitoringapplikationen und -Ansätze detektiert werden können, damit eine zeitnahe (manuelle oder automatische) Reaktion möglich wird. Zusätzlich wird darauf eingegangen, wie Versuche die Protokollierung zu umgehen oder zu stören, verhindert werden können. Die Ansprüche der digitalen Forensik werden vorgestellt und berücksichtigt.

Angestrebt wird, dass die Arbeit in verschiedenen Situationen/ Projekten grundlegende Hilfestellung bei dem Aufbauen eines Monitoring- und Loggingsystems geben kann. Es soll vereinfacht werden, ein der Detektion zuträgliches System zu entwerfen und eine Fokussierung auf wichtige Indikatoren eines Vorfalls zu verbessern.

In einem theoretischen Teil der Arbeit werden dazu zunächst die einzelnen Angriffe aus der von Microsoft entwickelten Bedrohungsmatrix analysiert. Die betrachteten Bedrohungen beschränken sich dabei auf die ausschließlich Kubernetes-spezifischen. Die folgende Struktur wird hierbei angewendet:

1. Bedrohung: Die Angriffe werden erläutert und motiviert. Es wird dargestellt, warum sie eine Gefahr für ein Kubernetes-System darstellen können und welche Ziele ein:e Angreifer:in zu erreichen versucht.
2. Indikatoren und allgemeine Maßnahmen: Es werden Indikatoren für Angriffe, die durch ein Logging-System feststellbar sind, gesucht und aufgeführt. Allgemeine Logging-basierte Detektionsmaßnahmen werden empfohlen.
3. Spezifische Maßnahmen: Zu jedem Angriff werden auch spezifische Logging- Maßnahmen zur Detektion aufgeführt.

Im Anschluss an den theoretischen Teil (Kapitel 3) wird für ausgewählte Fälle die Funktionstüchtigkeit des entwickelten Ansatzes und der aufgeführten Maßnahmen praktisch untersucht (Kapitel 4). Die Logging-Indikatoren respektive die identifizierten Maßnahmen sollen so evaluiert werden.

## 2 Grundlagen

### 2.1 Containerisierung & Kubernetes

Aktueller praktischer Stand ist es, Anwendungen möglichst zu kapseln und getrennt voneinander zu betreiben. Es soll eine einfachere Verwaltung der Anwendungen ermöglicht und aus Gründen der Sicherheit Grenzen zwischen Instanzen einer oder verschiedener Anwendungen geschaffen werden. Ein Weg, welcher Kosten einsparen kann, indem die Anzahl notwendiger physischer Komponenten reduziert wird, ist die Virtualisierung. Zu Beginn wurden Anwendungen i. d. R. direkt auf einem dedizierten physischen Server betrieben. Eine Abgrenzung wurde dadurch erreicht, dass je Anwendung ein Server genutzt wurde. Es wurde im Rahmen der Entwicklung von Virtualisierungstechnologien dazu übergegangen, physische Computer (vollständig) virtuell nachzubilden. „[J]ede VM [führt dabei] ihren eigenen Satz von Systemprozessen“ ([Luk18]) aus und ein sogenannter Hypervisor (z.B. VMWare oder VirtualBox) verwaltet die virtuellen Maschinen und ihre Hardwareressourcen. Dadurch konnten auf einem physischen Rechner mehrere Anwendungen auf eigenen virtuellen Rechnern betrieben werden. Da diese virtuellen Maschinen eng an die Eigenschaften des physischen Rechners gebunden sind (z. B. an die Architektur der CPU), die vollständige Virtualisierung viel Rechenleistung beansprucht und die Konfiguration sowie der Betrieb erhöhten Konfigurationsaufwand mitbringt, erfüllt sie die Ansprüche moderner Softwareentwicklung allerdings nicht ([Luk18]). Durch den hohen (Eigen-)Bedarf an Ressourcen der VM ist auch die Anzahl der Anwendungen, die betrieben werden können, stark limitiert. Eine leichtgewichtige Variante der Isolierung basiert auf den Linux-Containertechnologien. Auf Basis dieser lassen sich Anwendungen in sogenannte Container kapseln. Die Anzahl an Anwendungen, die auf einem physischen System betrieben werden kann, steigt dadurch enorm. Die Abbildung 2.1 veranschaulicht diese Entwicklung.

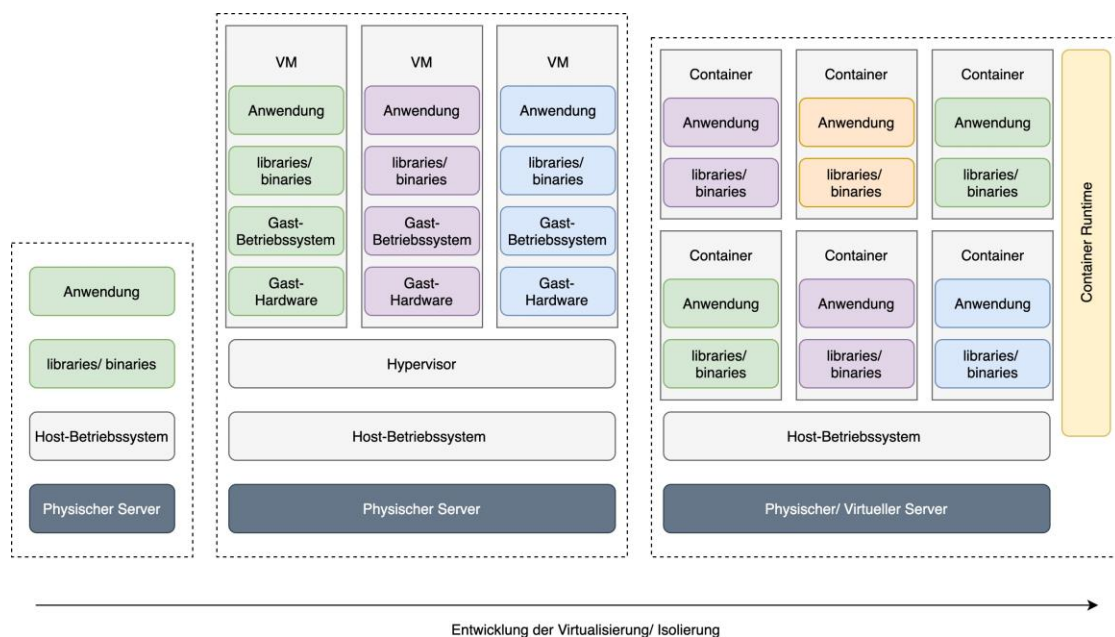


Abbildung 2.1: Entwicklung der Virtualisierung/ Isolierung

### 2.1.1 Containervirtualisierung

Der wesentliche Unterschied von Containern gegenüber virtuellen Maschinen ist die direkte Interaktion der Containerprozesse mit dem Host-Betriebssystem. Ein Container ist wie ein ‚normaler‘ Prozess auf dem Host zu verstehen. Um hier eine Isolierung zu erreichen, werden zwei wesentliche Mechanismen eingesetzt:

*Linux Namespaces* sind der Grundpfeiler für die Prozess-Isolierung. Sie „stellen sicher, dass jeder Prozess seine persönliche Sicht auf das System hat (Dateien, Prozesse, Netzwerkschnittstellen, Hostname usw.)“ ([Luk18]). Ein Namespace umfasst praktisch immer eine Art von Ressourcen. In einem modernen Linux-Betriebssystem gibt es einen Standardsatz von Namespaces. So z. B. der Netzwerk-Namespace (net), welcher die Netzwerkschnittstellen, die ein Prozess sieht und nutzen kann, verwaltet. Ein Prozess kann Teil von mehreren Namespaces sein, allerdings nur jeweils von einem Namespace einer Namespace-Art (also z. B. eines net-Namespaces) ([Luk18]). Wie in einem Baukastensystem kann ein Container dann durch die Namespaces mit Ressourcen bereichert werden und bleibt dennoch isoliert ([OR21]).

*Cgroups* werden genutzt, um die Ressourcen, die ein Container belegen bzw. maximal nutzen darf, zu beschränken. Die Abkürzung ‘Cgroups’ steht für ‘Control Groups’. Es handelt sich dabei um eine „Funktion des Linux-Kernels, die es ermöglicht, die Ressourcennutzung eines Prozesses (oder einer Gruppe von Prozessen) zu begrenzen“ ([Luk18]).

Um die Verwaltung von Containern zu erleichtern und vollständige Zusammenschlüsse von Konfigurationen, Dateien und Bibliotheken portabel zu machen, werden moderne ‘Container-Runtimes’ genutzt. Eine der bekanntesten Runtimes ist ‘Docker’ (Kern-Runtime ist hier ‘containerd’) [Doc21b]. Diese Zusammenschlüsse werden ‘Container- Images’ genannt und sind mit Abbildern virtueller Maschinen vergleichbar. Das Image umfasst die Anwendungsdateien und ihre Abhängigkeiten, „sowie andere Metadaten, z. B. den Pfad zu der ausführbaren Datei, die gestartet werden soll, wenn das Image ausgeführt wird“ ([Luk18]). Ein Image hat keinen Zustand und verändert sich nicht ([Doc21a]). Eine Besonderheit der Images ist der modulare Aufbau in Schichten. Jeder einzelne Schritt, der das fertige Image erzeugt, wird dabei in einer Schicht angelegt. Die Schichten zusammen ergeben das Image und könne auch von anderen Images wiederverwendet werden. Es kann so auch ein effizientes Caching aufgebaut und der Speicherbedarf massiv reduziert werden ([Doc21a]).

Ein Container wird dann durch die Runtime auf Basis der beschriebenen Isolierungsmechanismen und eines Images erzeugt. Der konkrete Container als laufende Instanz eines Images hat einen Zustand und ist veränderlich ([Doc21a]).

### 2.1.2 Kubernetes

In Unternehmen und Organisationen, die ihre containerisierten Anwendungen in großem Maßstab bereitstellen wollen und diese Anwendungen u.U. hochverfügbar sowie skalierbar erreichbar machen möchten, bedarf es einer Hilfestellung, die (Betriebs-) Verantwortliche entlasten kann und ein zielgerichtetes Management ermöglicht [Luk18]. 2014 wurde Kubernetes (griech. Steuermann/ Pilot) von Google als Open-Source-Projekt veröffentlicht, welches die Herausforderungen der Bereitstellung von Anwendungen deutlich vereinfacht und deren einfache Skalierung ermöglicht. „Kubernetes ist eine [...] erweiterbare [...] Plattform für die Verwaltung von Container-Workloads und -Diensten [...]“ ([The21i]), die es auch erleichtert eine große Anzahl an physischen Rechnern (im Folgenden ‘Nodes’) in einem Verband (‘Cluster’) zu nutzen und die Arbeitslasten (in Form von Containern) darauf zu verteilen.

Kubernetes abstrahiert und gruppiert einzelne Container in Gruppen, den sogenannten ‘Pods’. Die Plattform stellt somit eine weitere Abstraktionsschicht oberhalb der Verwaltung einzelner

Container dar. Diese Pods können einen oder mehrere Container, die z. B. zu einer Anwendung gehören, umfassen und sind in Kubernetes die kleinste verwaltbare Einheit. Eine Kommunikation zwischen den Komponenten von Kubernetes, insbesondere der Pods, wird durch sogenannte Overlaynetzwerke realisiert. Diese Netzwerke sind hierarchisch flach und wird zwischen den Nodes virtuell aufgespannt. Neben den Pods gibt es noch andere Einheiten in Kubernetes, die an der grundsätzlichen Struktur in einem Cluster beteiligt sind. So gibt es z. B. sogenannte 'Services' („eine abstrakte Möglichkeit, eine Anwendung, die auf einer Reihe von Pods ausgeführt wird, als Netzwerkdienst verfügbar zu machen“ ([The21i])) oder 'Ingress-Objekte', um Services und Pods von außerhalb des Clusters erreichbar zu machen. Die Abbildung 2.2 veranschaulicht diese Struktur im Groben.

#### 2.1.2.1 Wichtige Komponenten von Kubernetes (spezifisch für Control Planes)

In einem Kubernetes-Cluster können Nodes zu zwei Typen gehören: zum einen zu den 'Control Planes', zum anderen zu den 'Worker Nodes'. Die Control Planes sind Nodes von denen aus das Cluster gesteuert und verwaltet wird. Es gibt einige Komponenten, die i. d. R. ausschließlich auf Control Plane Nodes ausgeführt werden. Diese werden im Folgenden kurz vorgestellt.

#### 2.1.2.2 Kube-API-Server

Die Kubernetes-API ist der zentrale Interaktionspunkt. Alle wesentlichen Aktionen und Konfigurationen erfolgen über diese Schnittstelle. Die verschiedenen Elemente von Kubernetes sind über API-Objekte abgebildet (z. B. Pods) und können über sie bearbeitet werden. Auch der Zustand der verschiedenen Objekte kann hier abgefragt werden [The21d]. Die zentrale Bedeutung des API-Servers führt entsprechend zu einem sehr hohen Schutzbedarf dieser Komponente. Die API folgt dem REST-Standard.

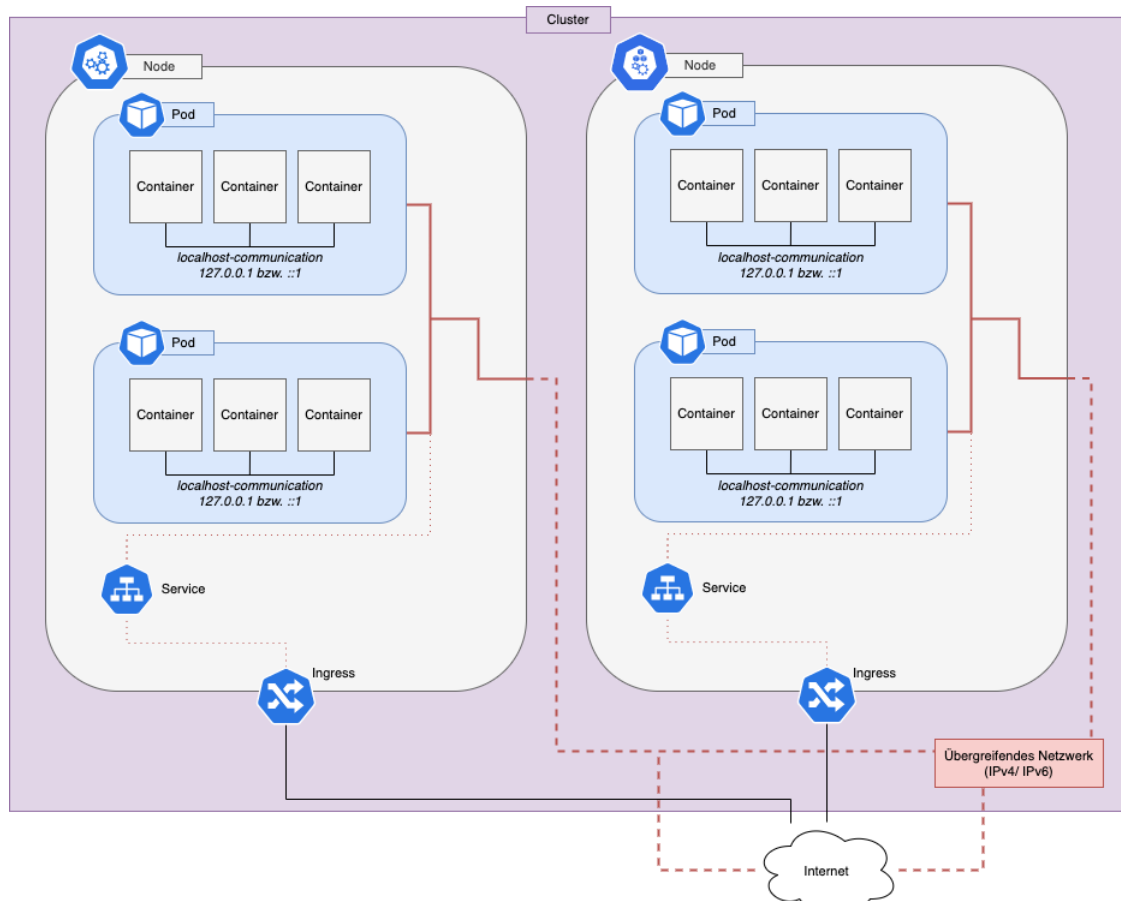


Abbildung 2.2: Grobe Struktur von Kubernetes

### 2.1.2.3 etcd

‘etcd’ ist der zentrale Speicher des Cluster-Zustands. In diesem „konsistenten und hochverfügbaren Schlüsselwertspeicher“ ([The21e]) werden die Zustände aller Komponenten und Objekte nachgehalten. Die Kubernetes-API greift auf den etcd-Speicher zu und kann Änderungen ablegen sowie die jeweiligen Zustände auslesen. Für den korrekten Betrieb des Clusters ist es wesentlich, dass der Zustand konsistent gespeichert ist. Die Kubernetes-API kann skaliert werden und der etcd-Speicher kann bei hochverfügbaren Systemen ebenfalls mit mehreren Instanzen betrieben werden.

### 2.1.2.4 Kube-Scheduler

Wenn ein neuer Pod erzeugt wird, muss dieser einem Node zugewiesen werden. Hierfür ist der ‘Kube-Scheduler’ verantwortlich. Unter Berücksichtigung von Kriterien wie zur Verfügung stehenden sowie angefragten Ressourcen, Beschränkungen durch Richtlinien oder Affinitäts- und Anti-Affinität-Spezifikationen wird entschieden auf welchem Node der Pod gestartet wird ([The21e]).

### 2.1.2.5 Kube-Controller-Manager

Der Kube-Controller-Manager ist eine Komponente, welche einen Satz an Steuerungsprozessen ausführt. Einige Arten von solchen Steuerungsprozessen sind:

- 1 Node-Controller: Verantwortlich für das Erkennen und Reagieren, wenn Nodes ausfallen.
- 2 Job-Controller: Achtet auf Job-Objekte, die einmalige Aufgaben darstellen und erstellt dann Pods, um diese Aufgaben zu Ende zu führen.
- 3 Endpoint-Controller: Befüllt das Objekt ‘Endpoints’ (d. h. er verbindet Dienste und Pods).
- 4 Service Account und Token Controller: Erstellt Standardkonten und API-Zugriffstoken für neue Namespaces.

*Aufzählung übersetzt nach [The21e]*

### 2.1.2.6 Wichtige Komponenten von Kubernetes (Allgemein & spezifisch für Worker Nodes)

Worker Nodes sind solche Rechner, auf denen die eigentliche Arbeitslast ausgeführt wird. Auf Control Planes wird i. d. R. keine Arbeitslast ausgeführt, um eine Gefährdung der zentralen Steuerungskomponenten zu minimieren und genügend Ressourcen für diese vorzuhalten. Es gibt einige Komponenten, die auf jedem Node im Cluster ausgeführt werden. Diese werden im Folgenden kurz vorgestellt.

### 2.1.2.7 kubelet

Diese Komponente wird auf jedem Node des Clusters ausgeführt und ist die Node-lokale Steuerungseinheit für Pods und Container. Der Lebenszyklus eines Pods und seiner Container wird von kubelet betreut und umgesetzt. „Kubelet verwaltet keine Container, die nicht von Kubernetes erstellt wurden“ ([The21e]).



### 2.1.2.8 kube-proxy

Der 'kube-proxy' ist eine Netzwerk-Komponente, die wie 'kubelet' auf jedem Node ausgeführt wird. Kubernetes nutzt in seiner Netzwerkstruktur eine Service-Struktur. Hierbei können Service-Objekte erzeugt werden, um Pods und ihre Dienste im Cluster über das Netzwerk verfügbar zu machen. Der kube-proxy realisiert diese Struktur durch Nutzung des Paket-Filters (z.B. iptables bei Linux) des Nodes. Der kube-proxy stellt auch eine Brücke zwischen den Cluster-internen und -externen Netzwerken dar. Dienste können über diese Komponente auch von außerhalb des Clusters erreichbar gemacht werden und Verkehr von außen kann Dienste im Cluster erreichen ([The21e]).

### 2.1.2.9 Netzwerk-Komponenten

In einem Kubernetes Cluster gibt es viele verschiedene Komponenten des Netzwerks. Einige mit besonderer Relevanz sind die Folgenden:

- **Netzwerk-Plugin:** Im Rahmen des Kubernetes Projektes, wurde entschieden, die Verwaltung des übergreifenden Netzwerkes nicht selbst zu lösen, sondern durch das Bereitstellen einer standardisierten Schnittstelle ('Container Network Interface' (CNI)) offen zustellen, wie und mit welcher Software (sogenannte CNI-Plug-ins) das Netzwerk aufgebaut und verwaltet werden soll. Diese Plug-ins sind Implementierungen des Kubernetes-Netzwerkmodells, die sich in den netzwerktechnischen Konzepten und in Zusatzfunktionen unterscheiden. Sie müssen das eigentliche Netzwerk der Pods (inklusive des IP-Adressenmanagements (IPAM)) bereitstellen, sowie das Node übergreifende Netzwerk aufbauen. Da sie beliebig erweiterbar sind, ermöglichen sie oft auch die Verwaltung von weiteren Endpunkten und Elementen wie z. B. das Definieren von Netzwerkregeln.

Das grundlegende Netzwerkmodell von Kubernetes verlangt Folgendes:

- Jeder Pod kann mit jedem Pod kommunizieren, ohne dass eine Übersetzung der Adressen (NAT bei IPv4, IETF RFC 2663) notwendig ist. Dies gilt über alle Nodes (Rechner in einem Cluster) des Clusters hinweg. ([The21l])
- Die verwaltenden Programme von Kubernetes auf jedem Node (z. B. kubelet) müssen mit jedem Pod auf dem 'eigenen' Node kommunizieren können. ([The21l])
- Sollten Pods in dem Host-Netzwerk liegen, müssen sie trotzdem mit allen anderen Pods auf jedem Node ohne NAT (bei IPv4) kommunizieren können. ([The21l])
- IP-Adressen werden nur einem Pod zugewiesen. Die Container in einem Pod teilen sich eine IP- (Layer-3, Network Layer, IPv4/ IPv6) und MAC-Adresse (Layer-2, Data Link Layer, IEEE 802 LAN Standards). Die Erreichbarkeit der Container untereinander erfolgt über die Adressierung von Ports des Ziels 'localhost'. Dies führt dazu, dass Container eines Pods nicht die gleichen Ports belegen dürfen. ([The21l])
- **Cluster-DNS:** Durch Domain-Name-Services (DNS) werden namentliche Adressen (z.B. microservice-a.firma.svc.cluster.local) in IP-Adressen übersetzt. „Cluster-DNS ist ein DNS-Server, der DNS-Einträge für Kubernetes-Dienste bereitstellt. Von Kubernetes gestartete Container beziehen diesen DNS-Server automatisch in ihre DNS-Suche ein“ ([The21e]). So können z.B. Pods bzw. ihre Dienste über DNS-Namen im Cluster erreichbar gemacht werden.



## 2.2 Logging & Monitoring

Wenn Anwendungen und Systeme betrieben werden, besteht von verschiedenen Seiten ein Interesse daran, bestimmte Ereignisse in diesen Anwendungen und diesen Systemen zu erfassen.

Nach A. Chuvakin, K. Schmidt und C. Phillips [CSP13] gibt es drei wesentliche Gründe für das Erfassen von Ereignissen:

### **Sicherheit**

Der Fokus liegt hier auf der Erkennung von Bedrohungen für das System respektive dessen Daten.

### **Betrieb & Wartung**

Ziel ist es, den korrekten Betrieb des Systems zu überwachen und auf veränderte Zustände reagieren zu können. Auch fehlererkennende Maßnahmen, die das Entwicklerteam über Probleme informieren oder den Betrieb optimieren (z. B. Ressourcennutzung), sind hier relevant.

### **(Nachweis der) Einhaltung von Vorschriften**

Für einige Anwendungsfälle von IT-Systemen gibt es Vorschriften, die zwingend eingehalten werden müssen (z. B. im Finanzbereich). Einige dieser Vorschriften erzwingen Maßnahmen zur Ereigniserfassung oder dass der Nachweis über die Einhaltung von bestimmten Maßnahmen geführt werden muss.

### 2.2.1 Definitionen

Das Prinzip des Loggings wird genutzt, um die oben genannten Ziele zu erreichen. Um eine zielgerichtete, konsistente und maschinell gut umsetzbare Verarbeitung von Ereignissen zu erreichen, sollten Log-Daten folgende Fragen beantworten (*nach [CSP13] und [PF21]*):

- Was ist passiert: Ereignistyp und Kategorie
- Wann ist es passiert: Zeitstempel
- Wo ist es passiert: z. B. System oder Prozess
- Wer war der Auslöser: z. B. Nutzernamen oder IP-Adresse
- Welche Parameter wurden verwendet: z. B. HTTP-Request-Parameter
- Wie sah das Ergebnis aus: z. B. erfolgreich oder Zugriff verweigert
- Wer hat das Ereignis gemeldet: z. B. Intrusion-Detection-System

Ein 'Event' (Ereignis) beschreibt dabei das (einzelne) Auftreten einer Zustandsänderung und seines Kontexts (Zeit, Details der Ursache, etc.). Der Kontext wird durch einzelne, vom jeweiligen System definierten 'Event-fields' abgebildet. Beispiele hierfür sind Datum, Uhrzeit, Nutzernamen oder 'Node-Kennung'. Der Zusammenschluss von Event-fields ist ein 'Event-record' (oder auch 'Log-Eintrag'). Die Sammlung von Event-records z. B. in einer Datei ist ein 'Log'. Der Prozess des Sammelns von Event-records in Logs wird 'Logging' genannt ([The10] und [PF21]).

Die Abbildung 2.3 verdeutlicht diese Zusammenhänge.

Logging ist in der Regel ein automatisierter Prozess. Entsprechend der programmatischen Umsetzung und der Konfiguration von Systemen bzw. Anwendungen erzeugen diese selbstständig und fortlaufend Event-records. Diese Einträge können dabei auf verschiedene Art und Weise strukturiert sein ([PF21]):

### Unstrukturiert

*Accepted password for root*

### Semi-strukturiert

*2021-01-05T13:37:00 Accepted password for root*

### Strukturiert

*timestamp="2021-01-05T13:37:00" action="login" user="root"*

Um eine automatische weiterführende Verarbeitung und Analyse zu ermöglichen, sollten Event-Records in strukturierter Weise angelegt werden. Durch die gleichbleibende Struktur und das spezifische Ausweisen von Event-fields (z.B. 'timestamp=...') kann ein anderes System diese Felder erkennen und verarbeiten. Bei unstrukturierten Einträgen ist dies nur schwer möglich, da nicht in gleichbleibender und eindeutiger Weise auf Informationen zugegriffen werden kann.

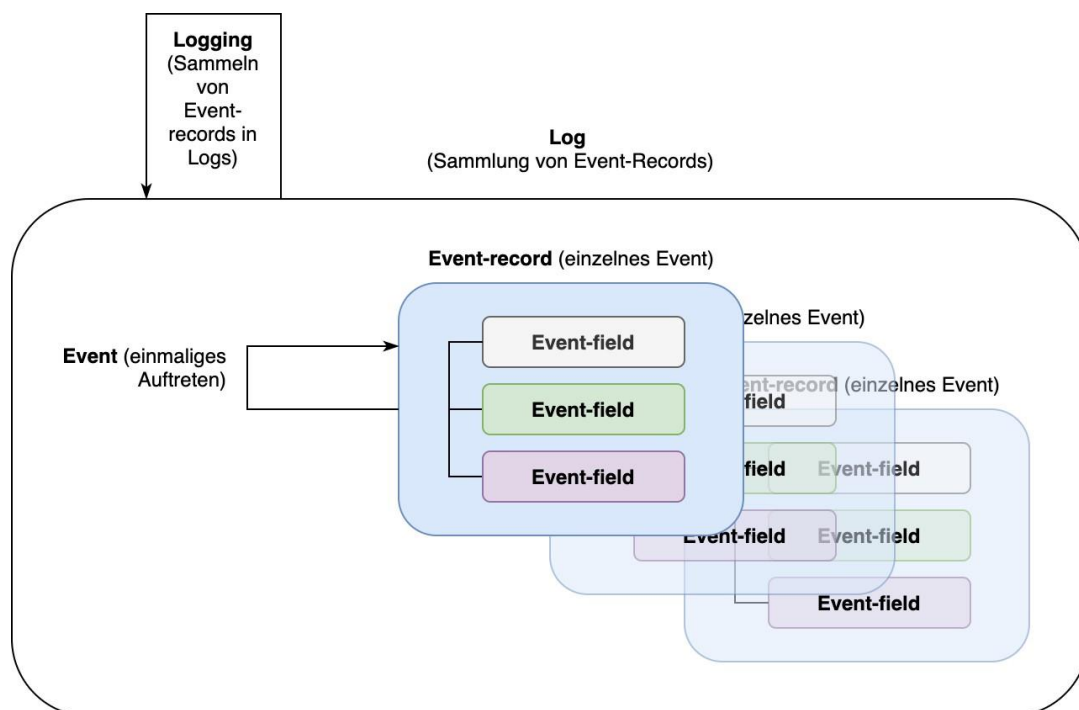


Abbildung 2.3: Definition Log

## 2.2.2 Dezentrales und zentralisiertes Logging

Es gibt zwei Architekturen, in denen Logging-Systeme realisiert werden. Die genaue Ausgestaltung und Komplexität kann stark variieren und hängt vorrangig von den Anforderungen an das System ab ([KSN06]).

Unter der Annahme, dass jedes System sowie jede Anwendung in einer Infrastruktur Log-Einträge erzeugt, liegt es nahe, wenn diese zunächst auf dem eigenen System abgelegt werden. Bei Anwendungen bietet es sich an, diese z. B. an das Betriebssystem zu übergeben. Diese unabhängige Speicherung wird als dezentrales Logging bezeichnet (siehe auch Abbildung 2.4). Jedes System ist für die Verwaltung und Verarbeitung von Logs zuständig. Ein:e Administrator:in kann an jedem System der Infrastruktur die jeweiligen Log-Daten des Systems einsehen und eventuelle Auswertungen oder Alarmmeldungen abrufen.

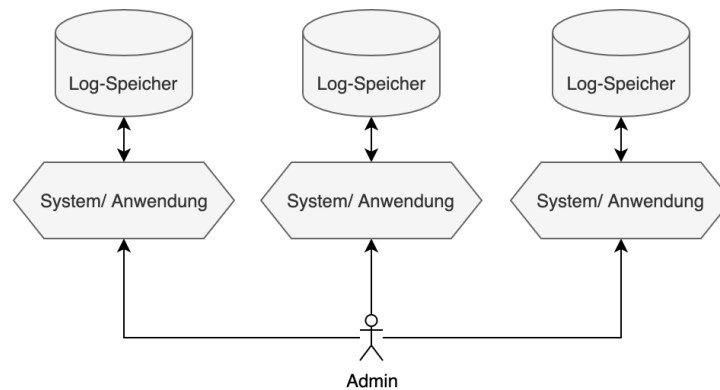


Abbildung 2.4: Dezentrales Logging

In großen Infrastrukturen wird dieses Vorgehen unpraktikabel, da mit steigender Anzahl von Systemen und Anwendungen in der Infrastruktur der Aufwand der Überwachung größer wird. Es muss ebenfalls berücksichtigt werden, dass in dieser Architektur weiterführende Auswertungen der Log-Daten auf die Daten eines Systems beziehungsweise einer Anwendung beschränkt sind. Eine übergreifende Verarbeitung z. B. eine Beobachtung von Ereignis-Ketten über verschiedene Systeme hinweg ist nicht möglich.

Die Architektur-Variante des zentralisierten Loggings ermöglicht einen vereinfachten Zugriff auf Log-Daten einer größeren Anzahl an Systemen und eine zusammenhängende Auswertung. Bei dem zentralisierten Logging gibt es einen dedizierten Logging-Dienst respektive Server, der Log-Daten von den verschiedenen Systemen gesammelt speichert (siehe Abbildung 2.5).

Eine Übertragung der Event-Records erfolgt i. d. R. über Netzwerkverkehr und kann auf Basis zweier Prinzipien erfolgen:

### Push-Verfahren

Bei dem Einsatz des Push-Verfahrens werden die Daten von dem Log erzeugenden System/ Anwendung oder einem Agenten (siehe agentenbasiertes Logging) an den zentralen Logging-Dienst gesendet.

### Pull-Verfahren

Wird das Pull-Verfahren genutzt, fragt der zentrale Logging-Dienst in regelmäßigen Intervallen bestimmte Endpunkte der Systeme, Anwendungen oder Agenten ab, um die Log-Daten einzusammeln.

Je nach eingesetztem Logging-System gibt es noch Variationen der für die Logs zuständigen lokalen Einheit. So kann ein spezifischer sogenannter (Logging-) 'Agent' eingesetzt werden. Dieser wird z. B. auf jedem Server einer Infrastruktur betrieben und sammelt dort die Logs aller Anwendungen auf dem Server und die des Servers selbst ein und sendet diese an den zentralen Logging-Dienst (Agent-based, vgl. [KSN06]). Je nach Infrastruktur wird dieser Agent auch spezifisch an eine Anwendung/ eine Systemkomponente gebunden und sammelt nicht umfassend Log-Daten ein. Schlussfolgernd ist es auch möglich Logging-Systeme einzusetzen, bei denen ein System selbst direkt Logs an den Logging-Dienst sendet ('Agentless' mit Push-Verfahren, vgl. [KSN06]).

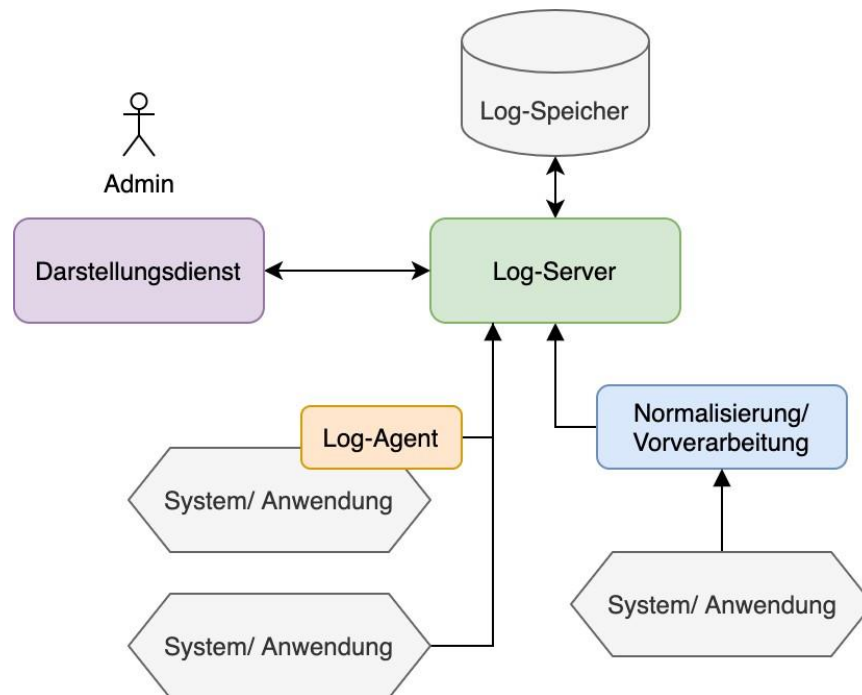


Abbildung 2.5: Zentralisiertes Logging

### 2.2.3 Sicherheit bei der Übertragung von Log-Daten

Bei der Verarbeitung und Übertragung von Log-Daten gilt es, grundlegende Sicherheitsmaßnahmen zu ergreifen. Je nach Einsatzumgebung können die Daten sensible Informationen enthalten und haben daher aus Datenschutzperspektive einen Schutzbedarf. Betrachtet man zusätzlich das oben genannte Ziel, die Sicherheit eines Systems zu verbessern, wird es notwendig Schutzmaßnahmen zu ergreifen. Ein:e Angreifer:in kann versuchen, Log-Daten zu stehlen, zu manipulieren oder gefälschte Log-Daten einzuspielen, um etwa den Diebstahl von Unternehmensdaten zu verschleiern.

In ihrer Veröffentlichung ‘Guide to Computer Security Log Management’ führt die NIST (National Institute of Standards and Technology, USA, [KSN06]) drei grundlegende Maßnahmen auf, die zur Absicherung der Logs und Wahrung von Schutzzielen umgesetzt werden sollten:

#### **Verlässliche Übertragung**

Bei der Übertragung über Netzwerke sollten Protokolle verwendet werden, die eine Zustellung in höherem Maße sicherstellen. So ist konkret das Transmission Control Protocol (TCP) gegenüber dem User Datagram Protocol (UDP) zu bevorzugen.

#### **Vertraulichkeit bei der Übertragung**

Um sicherzustellen, dass nur befugte Komponenten die übertragenen Daten lesen können, sollten Verfahren wie das Transport Layer Security (TLS) Protokoll verwendet werden. Auch der RFC 6587 empfiehlt den Einsatz von TLS bei dem Einsatz von syslog (Standardprotokoll, um Systemprotokoll- oder Ereignismeldungen an einen bestimmten Server zu senden (übersetzt nach [Pas21])).

#### **Integrität und Authentifizierung**

Da ein:e Angreifer:in es versuchen kann, Log-Daten zu manipulieren, empfiehlt sich der Einsatz von Maßnahmen zur Sicherung der Korrektheit und Überprüfung auf Veränderung. Kryptografische Hash-Funktionen können eingesetzt werden, um die

Integrität sicherzustellen. Bei Verbindungen mit TLS gibt es, in neueren Versionen und bei der Verwendung von bestimmten Ciphersuiten, die Möglichkeit die Vertraulichkeit und die Integrität gemeinsam sicherzustellen. Die NIST empfiehlt über die Sicherung der Integrität hinaus auch die Authentifizierung. Um zu erreichen, dass nur berechnete Komponenten z. B. Log-Daten an den zentralen Log-Server senden dürfen, können Authentifikationsverfahren eingesetzt werden (ein einfaches Beispiel wäre eine Authentifizierung durch Nutzernamen und Passwort).

## 2.2.4 Elastic-Stack - Zentralisierte Logging-Architektur

Der Elastic-Stack ist eine Sammlung von Open-Source Anwendungen. Klassisch gehören zu ihm die Anwendungen 'Elasticsearch', 'Logstash' und 'Kibana' (auch 'ELK- Stack' genannt, [Ela21c]). Da für den Anwendungsfall dieser Arbeit 'Logstash' nicht relevant ist, wird diese Komponente nicht weiter berücksichtigt. Im Folgenden werden die in dieser Arbeit verwendeten Komponenten der Sammlung vorgestellt.

### 2.2.4.1 Elasticsearch-Datenbank

Die Elasticsearch-Datenbank ist der zentrale Speicher des Systems. Sie legt Datensätze in Indizes gruppiert in Form von JSON-Dokumenten ab. Dieses dokumentenorientierte Datenbankmodell ist grundlegend verschieden von einer relationalen Datenbank (Stichwort: SQL) bei der Daten in Tabellen und deren Zeilen strukturgleich abgelegt werden. Ein Index in der Elasticsearch kann als „optimierte Sammlung von Dokumenten betrachtet werden, wobei jedes Dokument eine Sammlung von Feldern ist, die Schlüssel- Wert-Paare umfassen, die die Daten enthalten. Standardmäßig indiziert Elasticsearch alle Daten in jedem Feld und jedes indizierte Feld hat eine eigene, optimierte Datenstruktur“ ([Ela21b]). Sie ist besonders optimiert für große Datenmengen die aus verschiedenen Quellen stammen, Struktur-verschieden sind und einfach durchsuchbar sein sollen ([Ela21b]).

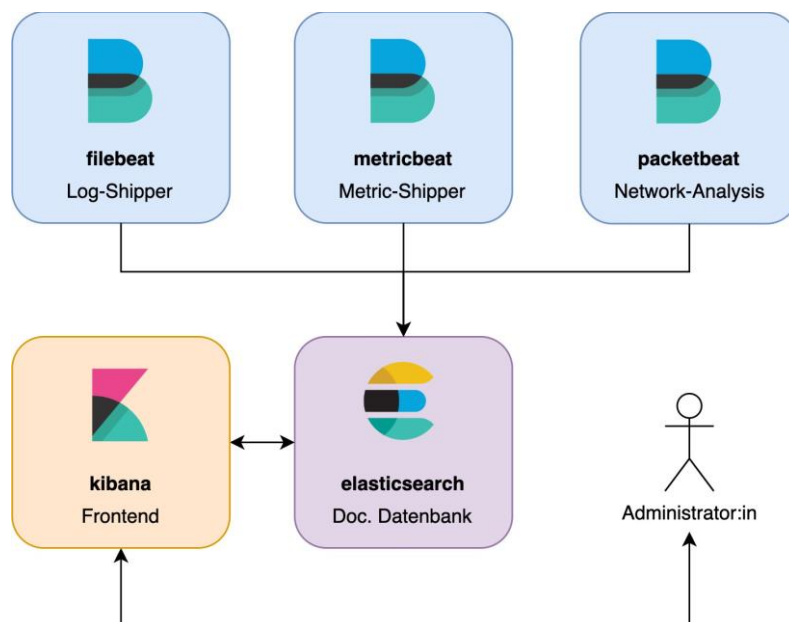


Abbildung 2.6: Schematische Struktur des Elastic-Stacks

#### 2.2.4.2 Kibana

Kibana ist eine webbasierte Anwendung zur Visualisierung, Untersuchung von Daten und einfachen Interaktion mit einer Elasticsearch-Datenbank. Auch Verwaltungsschritte wie das Anlegen von Indizes in der Elasticsearch-Datenbank lassen sich über Kibana durchführen ([Ela21e]). Diese Anwendung ist als zentraler Interaktionspunkt für Akteure, wie z. B. Administrator:innen zu verstehen.

#### 2.2.4.3 Beats

Unter der Bezeichnung ‘Beats’ werden die Anwendungen zusammengefasst, welche Daten aus verschiedenen Quellen an die Elasticsearch-Datenbank übermitteln.

In dieser Arbeit werden die folgenden Beat-Anwendungen eingesetzt:

##### **Filebeat**

‘Filebeat’ ist ein ‘Shipper’ (Agent der Log-Daten verarbeitet und überträgt) für die Weiterleitung und Zentralisierung von Protokolldaten. Filebeat wird als Agent installiert und überwacht festgelegte Protokolldateien oder Speicherorte, sammelt Protokollereignisse und leitet sie an die Elasticsearch-Datenbank zur Indexierung weiter ([Ela21d]). In einem Kubernetes-Cluster kann Filebeat als ‘Daemonset’ (Kubernetes Ressource) installiert und ausgeführt werden. Bei entsprechender Konfiguration lässt es sich auf diese Weise leicht umsetzen, dass auf jedem Node des Clusters eine Filebeat Instanz ausgeführt wird. In einer typischen Kubernetes-Konfiguration, sowie auch in dieser Arbeit sendet Filebeat primär Log-Daten aus dem Verzeichniss `/var/log/containers/*.log` an die Elasticsearch-Datenbank.

##### **Metricbeat**

„Metricbeat ist ein Shipper, welcher regelmäßig Metriken vom Betriebssystem und von den auf dem Server laufenden Diensten sammeln kann. Metricbeat nimmt die gesammelten Metriken und Statistiken und sendet sie an die Elasticsearch- Datenbank“ (übersetzt nach [Ela21f]). Primär sammelt ‘Metricbeat’ Daten zu Ressourcen-Nutzung (z. B. CPU-Auslastung) oder z. B. Durchsatzraten bei bestimmten direkt auf dem Host ausgeführten Anwendungen. Ebenso wie Filebeat bietet es sich an Metricbeat in einem Kubernetes Cluster, als Daemonset auszuführen. Durch Einbindung des Pfades `/proc` kann Metricbeat Daten über die System-Metriken eines klassischen Linux-Nodes sammeln.

##### **Packetbeat**

‘Packetbeat’ ist ein Echtzeit-Netzwerk-Paket-Analysator, der mit der Elasticsearch-Datenbank verbunden werden kann, um ein System zur Anwendungsüberwachung und Leistungsanalyse bereitzustellen. Packetbeat vervollständigt die Beats-Sammlung, indem es die Überwachung des Netzwerkverkehrs zwischen den Servern eines Clusters ermöglicht. Der Dienst lauscht an den Netzwerkschnittstellen des Nodes, analysiert verschiedene Protokolle auf Anwendungsebene und setzt die Nachrichten zu Transaktionen in Beziehung. Auch Packetbeat kann als Daemonset ausgeführt werden, um eine Ausführung auf allen Nodes zu erreichen. Den Instanzen der Anwendung muss die Capability `NET_ADMIN` zugewiesen sein und eine Einbindung in das Host-Netzwerk muss erlaubt sein. (nach [Ela21g])

#### 2.2.4.4 Backpressure Awareness

Das verwendete Protokoll zur Kommunikation zwischen den Beats und der Elasticsearch-Datenbank ist 'backpressure aware'. Das bedeutet, sollte die Elasticsearch-Datenbank oder andere Komponenten nicht mehr erreichbar sein, oder die Menge an übermittelten Daten für die Verarbeitung in der Elasticsearch-Datenbank zu viel sein, hält die Beat-Instanz die erfassten Daten zurück. Sobald der Normalzustand wieder erreicht wurde, werden die Daten übermittelt. Realisiert wird diese Funktion durch interne 'Queues'. Diese sind wie Warteschlangen zu verstehen. (nach [Ela21a])

#### 2.2.5 Falco - Kubernetes threat detection engine

Falco ist ein Open-Source-Projekt, welches von der Cloud Native Computing Foundation (CNCF) verwaltet wird. Das Projekt definiert sich selbst als Sicherheits- Tool welches potenzielle Bedrohungen zur Laufzeit erkennen kann. Falco liest und verarbeitet dazu einen ausgewählten Teil der Linux-System-Aufrufe (syscalls). Das sind Aufrufe von Kernel-Funktionen (wie z. B. Lesen oder Schreiben von Dateien oder das Erzeugen eines Prozesses) die aus dem User-Space (der Systembereich der z. B. vom Nutzer ausgeführte Anwendungen umfasst) erfolgen ([com21] und [Ric17]). Bildlich setzt sich Falco also zwischen die Anwendungen von Nutzern des Systems (inkl. Container) und den 'tieferen' respektive privilegierten Ebenen des Betriebssystems (Kernel). Der Strom an Linux-System-Aufrufen wird von Falco gegen definierte Regeln geprüft und im Fall eines Verstoßes wird eine Alarmierung ausgelöst.

Falco kann so das Verhalten des Kernels auf ungewöhnliches Verhalten hin untersuchen. Das umfasst unter anderem:

- Privilegienerweiterung durch privilegierte Container
- Lesen/Schreiben in Verzeichnissen wie /etc, /usr/bin, /usr/sbin, etc.
- Eigentümerschaft- und Moduswechsel
- Ausführen von Kommandozeilenprogrammen wie sh, bash, csh, zsh usw.
- Mutation von shadowutil- oder passwd-Programmen wie shadowconfig, pwck, chpasswd, getpasswd, change, useradd usw. und anderen
- Unerwartete Netzwerkverbindungen oder Socket-Mutationen
- Und Weitere...

In der Standardkonfiguration liefert Falco schon eine umfassende Menge an Regeln mit, um Bedrohungen und auffälliges Verhalten zu erkennen. Diese Regeln lassen sich ergänzen und abändern, um z. B. Ausnahmen hinzuzufügen.

In dieser Arbeit werden an verschiedenen Stellen Falco-Regeln hervorgehoben und erklärt. Die angenommene Betriebsweise von Falco ist hier die Ausführung als Daemonset. Falco kann über verschiedene konfigurierbare Wege Alarmmeldungen zustellen. In dieser Arbeit wird vor allem im praktischen Teil (Kapitel 4.1.2) erläutert, wie und warum die ausgewählten Kommunikationswege verwendet werden. Betrachtet wird der Aufruf von Webhooks eines Messaging-Dienstes und das Senden von Ereignissen an die Elasticsearch-Datenbank.

*Vorstehender Abschnitt nach: [TT21c], [TT21a] und [TT21b].*



## 3 Konzept

### 3.1 Identifizierung relevanter Störungen und Angriffe

Mit dem immer häufigeren Einsatz von Containerisierungsumgebungen und Orchestrierungslösungen wie Kubernetes, wird es zunehmend wichtiger diese Systeme sicher zu betreiben. Nach einer Untersuchung der Firma Red Hat wird Kubernetes mit einem Anteil von 87% bzw. 75% in Produktion bei den befragten Unternehmen<sup>1</sup> als Orchestrierung-System eingesetzt ([Red21]). Für Angreifer:innen wird es somit zunehmend lohnender Systeme auf Basis von Kubernetes anzugreifen.

Schwachstellen in Kubernetes sind entsprechend kritisch. So meldete z.B. im Dezember 2020 die für Sicherheit zuständige Kubernetes-Entwicklergruppe, eine kritische Lücke. „Nutzer, die mit entsprechenden Rechten im Cluster ausgestattet sind, [können] einen Container im Cluster starten und auf diesen Verkehr umleiten [...], der für externe Adressen bestimmt ist“ ([Mah20]). Ein:e Angreifer:in könnte so kritischen Netzwerkverkehr abfangen und auslesen.

In Kubernetes gibt es zwei wesentliche Arten von Schwachstellen bzw. Gründe für Schwachstellen: solche programmatischen Ursprungs (siehe Beispiel oben) und solche, die durch fehlerhafte Konfigurationen entstehen. Die Relevanz der beiden Ursprünge hat die Cloud Security Alliance untersucht und kommt zu dem Ergebnis, dass die falsche Konfiguration von Systemen deutlich häufiger der Grund für erfolgreiche Angriffe ist. Die Cloud Security Alliance führt diesen Grund als zweitkritischste Bedrohung für Cloud Systeme auf ([CBB<sup>+</sup>20]). Falsche Konfigurationen können Angriffe von innen und außen ermöglichen. Häufig werden automatisierte Verfahren genutzt, um z. B. versehentlich veröffentlichte Zugangsdaten [JRA14] zu finden und auszunutzen. Aber auch Angriffe, bei denen auf die versehentliche Fehlkonfiguration durch einen Administrator gehofft wird, z. B. bei dem Namen eines Container-Images (z. B. ‘tensorflow’ vs. ‘tensroflow’) oder bei denen öffentlich verfügbare Images direkt manipuliert werden, stellen eine große Bedrohung dar (vgl. [Vaa21] oder [Goo18]). Das Ziel von Angreifer:innen kann hier unterschiedlich ausfallen. So kann z. B. ein Ziel der Missbrauch von Ressourcen sein (z. B. Mining von Kryptowährungen) oder die Ausweitung von Rechten, um weitere Teile der Infrastruktur zu kompromittieren ([TD20]).

Angriffe von innen werden von der Cloud Security Alliance ebenfalls unter den zehn relevantesten Angriffen aufgeführt ([CBB<sup>+</sup>20]). Die Bedrohung geht dabei von Mitarbeiter:innen aus und hat oft Sabotage zum Ziel. Die National Security Agency (USA) identifiziert in ihrem ‘Kubernetes Hardening Guide’ solche Angriffe durch Mitarbeiter:innen als relevante Bedrohungen ([NC21]). Die Komplexität und Größe eines Kubernetes-Systems erhöht die mögliche Fläche für Angriffe. Die oben beispielhaft genannten Angriffsarten und -wege sollten verdeutlichen, dass es eine große Vielzahl von möglichen und unterschiedlichen Angriffen auf ein Kubernetes-System gibt. Um im Sinne des Ziels dieser Arbeit, diese Angriffe zu analysieren und Indikatoren zur Erkennung durch den Einsatz von Logging-Systemen zu finden und aufzuzeigen, bedarf es eines systematischen Vorgehens.

<sup>1</sup>Befragte vornehmlich aus der ‘Technologie-’ und ‘Finanzdienstleistungs-’ Branche, 491 Unternehmen wurden befragt



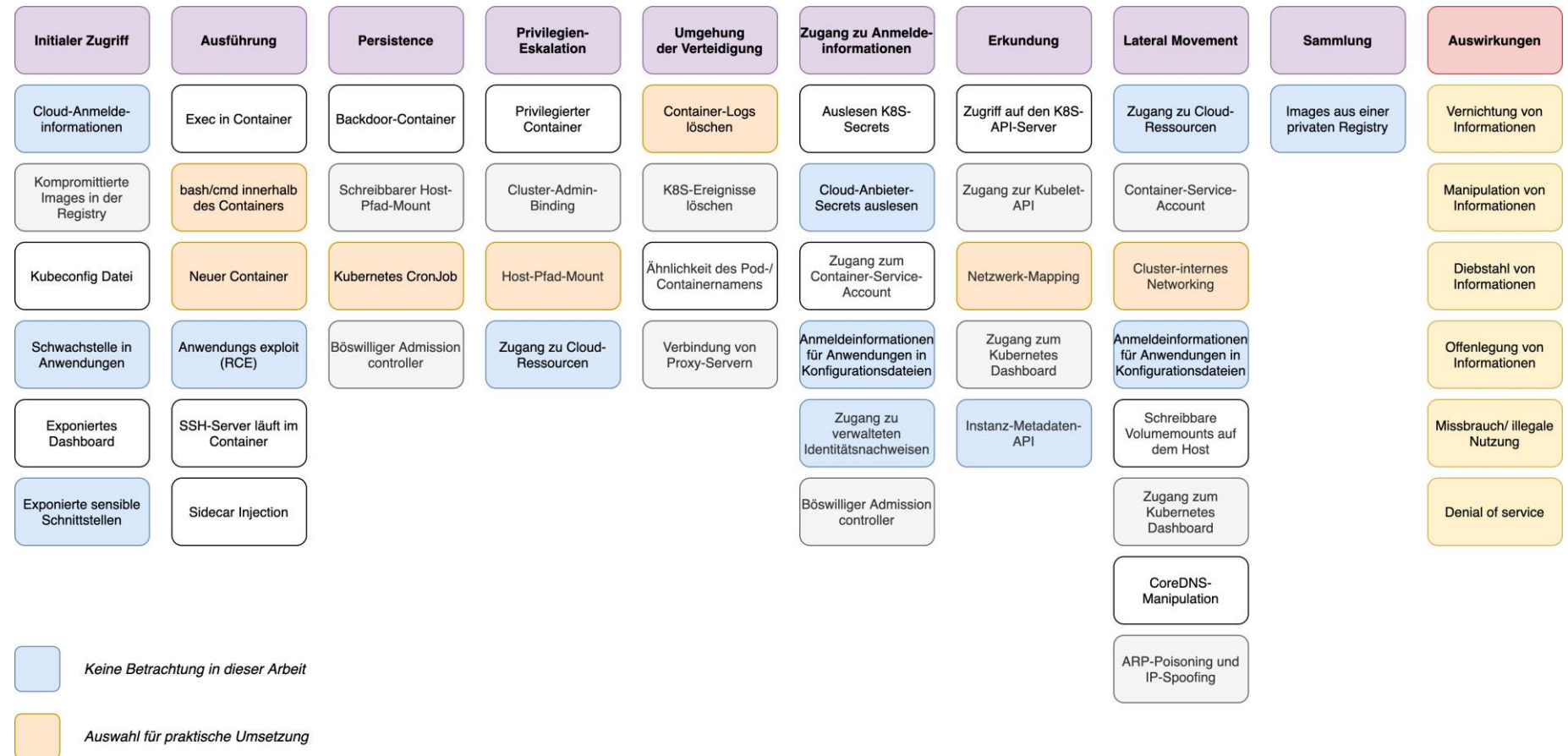


Abbildung 3.1: Bedrohungsmatrix für Kubernetes (nach [WM21])

Die MITRE ATT&CK Matrix ([The21j]), ein Framework zur Klassifizierung und systematischen Organisation von „Taktiken, Techniken und Verfahren (TTPs<sup>2</sup>) von Cyberkriminellen“ ([Lub]), ist ein bewährtes Mittel, um Angriffe auf eine IT- Infrastruktur zu untersuchen und bei der IT-Sicherheitsplanung Lücken zu vermeiden. Das Unternehmen Microsoft, welches als Public Cloud Provider selber Kubernetes Dienste anbietet, hat die Methodik der MITRE ATT&CK Matrix in eine Kubernetes spezifische Matrix übertragen ([WM21], Abbildung Figure 3.1).

Die Matrix beschreibt 9 Taktiken (lila gefärbte Felder, Abbildung Figure 3.1) von Angreifer:innen mit dazugehörigen Angriffs-Techniken. Die Taktiken wurden dabei von der MITRE ATT&CK Matrix ([The21j]) übertragen und bilden die wesentlichsten Angriffsziele ab. Zu den Taktiken wurden häufige Techniken zugeordnet, die jeweils Andeutungen zu möglichen Umsetzungswegen geben. Die Betrachtung möglicher Auswirkungen wurde präzisiert auf Basis der Klassifizierungen von Sicherheitsbedrohungen in Informationssystemen von M. Jouini et al. ([JRA14]).

Im Folgenden werden die aufgeführten Techniken theoretisch untersucht. Es wird eine Operationalisierung vorgenommen und allgemeine sowie spezifischere Hinweise zu einer Erfassung durch ein Logging- / Monitoringsystem gegeben (vgl. 1.3 Ziel). Die in der Abbildung Figure 3.1 blau markierten Techniken werden im Weiteren nicht weiter betrachtet. Sie gehen entweder über den Allgemeinheitsanspruch in Bezug auf Kubernetes hinaus, da sie nur für Public Cloud Umgebungen relevant sind oder abhängig von betriebenen Anwendungen sind.

---

<sup>2</sup>engl., Tactics, Techniques and Procedures

## 3.2 Operationalisierung, Identifizierung von Indikatoren und Empfehlung von Maßnahmen

Die folgenden Untersuchungen der Bedrohungen und insbesondere die aufgeführten Maßnahmen erfolgen unter der folgenden Einschränkung: Aus den drei großen Feldern der Maßnahmen der IT-Sicherheit (Prävention, Detektion und Reaktion) wird ausschließlich die Detektion, hier geschärft auf Logging-Maßnahmen, betrachtet. Zur Mitigation der Bedrohungen ist es unerlässlich, dass aus anderen Bereichen ebenfalls Maßnahmen ergriffen werden. Diese können in bestimmten Fällen besser geeignet sein, um die aufgeführten Angriffe zu adressieren. Sie sind allerdings nicht Bestandteil dieser Arbeit.

### 3.2.1 (IZ) Initialer Zugriff

#### 3.2.1.1 (IZ.1) Kompromittierte Images in der Registry

##### **Bedrohung**

Images sind die Basis für einzelne Container-Instanzen. Sämtliche Workloads in einem Kubernetes-Cluster basieren auf diesen Images. Entsprechend sorgfältig sollten die Images ausgewählt werden (siehe Grundlagen Kubernetes 2.1.2). Ein:e Angreifer:in versucht hier über die Manipulation von Images Zugriff zum Cluster erlangen. Zwei wesentliche Angriffspfade gilt es zu beachten ([WM21]):

- *Private Container-Registries:* Viele Organisationen nutzen in ihren Infrastrukturen eigene oder angepasste Images. Diese werden i. d. R. in privaten Registries (Speicherort für Container-Images) abgelegt und von dort bei Bedarf abgerufen (*pull*). Sollte es zu einer Kompromittierung dieser Registry kommen, könnte ein:e Angreifer:in die dort liegenden Images manipulieren.
- *Öffentliche Container-Registries:* Der andere wesentliche Ort, von dem Images in die Infrastruktur einer Organisation geladen werden, sind öffentliche Registries wie der 'Docker-Hub'. Hier besteht die Gefahr, dass zum einen die dort liegenden als offiziell markierten Images (z. B. von einem Nginx) aufwendig kompromittiert wurden und eine Gefahr darstellen. Diese Variante ist allerdings für eine:n Angreifer:in schwer umsetzbar, da viele Projekte durch ihre Open-Source Struktur eine große Community an Entwicklern haben, die getäuscht werden müssen. Eine Variante die erfolgreicher ist und über die es eine Vielzahl an Berichten gibt (s. z. B. [RT20]) basiert auf Täuschung der Endnutzer. Ein:e Angreifer:in versucht hier durch ähnliche Benennung von Images und die ähnliche Gestaltung der öffentlichen Registry-Webseite die mögliche Verwechslung oder z. B. Tippfehler eines Administrators auszunutzen.

##### **Allgemeine Maßnahmen**

Allgemein wird empfohlen, dass Images, die im Cluster verwendet werden, im laufenden Betrieb überwacht werden und die Nutzung (Pulls, Erstellung von Containern) in einem Logging-System nachgehalten wird. Dadurch können ungewollte Images im System erkannt werden und im Nachgang eines erkannten Angriffs kann nachvollzogen werden, was für ein Image von der/dem Angreifer:in verwendet wurde.

### Spezifische Maßnahmen

Konkreter kann den genannten Angriffspfaden mit einem Logging in Verbindung mit kryptografischen Hash- und Signaturfunktionen entgegengewirkt werden. Es kann eine Hashsignatur des Images gebildet werden, um es eindeutig identifizieren zu können (vgl. Open-Source-Projekt 'kube-notary' [Kub21]). Auch die Metadaten der Container-Runtime (Image-Name und Versions-Tag) und die von Kubernetes erstellten und erfassten Ereignisse (z. B. beim Erstellen eines Pods) können als Quellen für eine Überwachung genutzt werden. Wird ein nicht bekanntes oder erlaubtes Image im System erkannt, kann ein Event-Record einem Log hinzugefügt werden, das dann z. B. in einem zentralen Logging-System aufgenommen wird, welches die Systemadministratoren alarmiert (Beispiel Alarmmeldung: *'Image <{ImageName}> wurde gepulled mit bisher nicht verwendetem Versions-Tag <{VersionsTag}>'*).

#### 3.2.1.2 (IZ.2) Kubeconfig Datei

##### Bedrohung

Die Kubeconfig-Datei (Beispiel siehe Listing 3.1) ist von zentraler Bedeutung für die Administration eines Clusters. In dieser Datei werden neben den allgemeineren Daten, wie Adressen der Control-Planes auch Zugangsdaten spezifiziert. Bei der Installation eines Kubernetes-Cluster mit dem Tool kubectl wird eine Kubeconfig-Datei automatisch im home-Verzeichnis des Nutzers erstellt und erlaubt vollen Zugriff auf das Cluster. Das Programm 'kubectl' (Kommandozeilen-Programm zur Verwaltung eines Kubernetes-Clusters) greift standardmäßig ebenfalls auf die Kubeconfig-Datei zurück, um zuzuordnen, an welchen Kubernetes-API-Server die Befehle gerichtet werden sollen und um sich gegenüber dem Kubernetes-API-Server zu authentifizieren.

Aufgrund der zentralen Bedeutung und den in der Regel hohen Rechten, die mit den enthaltenen Zugangsdaten erhalten werden können, ist eine Kubeconfig-Datei ein wichtiges Ziel für einen Angreifer:in.

```
1 apiVersion: v1
2 clusters:
3 - cluster:
4   certificate-authority-data: <{cert-data}>
5   server: https://{IP address | domain}>:6443
6   name: kubernetes
7 contexts:
8 - context:
9   cluster: kubernetes
10  user: kubernetes-admin
11  name: kubernetes-admin@kubernetes
12 current-context: kubernetes-admin@kubernetes
13 kind: Config
14 preferences: {}
15 users:
16 - name: kubernetes-admin
17   user:
18     client-certificate-data: <{cert-data}>
19     client-key-data: <{key-data}>
```

Listing 3.1: Beispiel Kubeconfig-Datei

Es werden hier zwei wesentliche Zugriffspfade, die ein:e Angreifer:in nutzen könnte, betrachtet:

- Wie oben beschrieben wird die Kubeconfig-Datei bei der Cluster-Erstellung automatisch auf dem aktuellen Host (bzw. im Cluster dann Control-Plane) abgelegt. Ein möglicher Pfad eines/ einer Angreifer:in verläuft also über den Host. Erlangt ein:e Angreifer:in Zugriff z. B. über einen nicht abgesicherten SSH- Zugang, dessen Nutzung für Wartung intendiert ist, könnte er/ sie Zugriff auf die Kubeconfig-Datei erlangen und infolgedessen auf das Cluster.
- Ein anderer möglicher Angriff aus der Virtualisierungs-Ebene heraus erfolgt über Container die privilegierten Zugriff auf ein Control-Plane haben oder einen Host- Pfad einbinden, welcher den Speicherort der Kubeconfig-Datei direkt umfasst oder z. B. das Anlegen von Host-CronJobs auf dem Control-Plane erlaubt (vgl. dazu PS.2 und PS.3).

## Allgemeine Maßnahmen

Um Zugriffe auf die Nodes erkennen zu können, wird es empfohlen, Log-Daten des Host-Betriebssystems ebenfalls in das zentrale Logging-System einzuspeisen. Daten zu Netzwerkverkehr können hier ebenfalls Hinweise auf anomales Verhalten/ Zugriffe liefern.

## Spezifische Maßnahmen

In klassischen Linux-Betriebssystemen werden Ereignisse wie z. B. SSH-Anmeldungen in Log-Dateien erfasst. Diese Dateien könnten z. B. mit der Filebeat-Anwendung in eine Elasticsearch-Datenbank, die als zentraler Log-Speicher in einem Logging-System dient, übertragen werden. Einige Anwendungen, wie die unter Ubuntu häufig verwendete, auf iptables basierende 'Uncomplicated Firewall (UFW)', erzeugen zusätzlich eigenen Log- Dateien, die nützliche Informationen zur Erkennung liefern können. Um konkret Netzwerkverkehr auf Node-Ebene zu erfassen und zu analysieren, könnte auch

die Packetbeat-Anwendung verwendet werden, die ebenfalls mit einer Elasticsearch- Datenbank kompatibel ist.

```
Sep  8 14:38:44 tantive kernel: [9058399.551618] [UFW BLOCK] IN=eth1 OUT=
MAC=68:05:ca:89:ad:45:00:24:dc:41:9f:c0:08:00 SRC=199.195.253.174 DST=185.242.112.34 LEN=44
TOS=0x08 PREC=0x20 TTL=244 ID=54321 PROTO=TCP SPT=41952 DPT=22 WINDOW=65535 RES=0x00 SYN
URGP=0
```

Listing 3.2: Beispiel UFW-Log-Eintrag

### 3.2.1.3 (IZ.3) Exponiertes Dashboard

#### Bedrohung

Das Kubernetes „Dashboard ist eine webbasierte Kubernetes-Benutzeroberfläche. [Es kann verwendet werden], um [...] Cluster-Ressourcen zu verwalten“ ([The21c]). Das Dashboard ist also das Frontend der Kubernetes-API. Sollte durch Fehlkonfiguration oder Kompromittierung von Zugangsdaten ein Zugriff auf das Dashboard für eine:n Angreifer:in möglich sein, ist ein Zugriff auf die Kubernetes-API möglich.

#### Allgemeine Maßnahmen

Da über das Kubernetes-Dashboard Anfragen an die Kubernetes-API gesendet werden, sollten hier die Logs der API überwacht werden, um unbefugte Zugriffe, die z. B. zeitlich nicht zu Aktionen von Mitarbeitern passen oder die wesentlich den Cluster-Zustand verändern zu erkennen.

Um den kritischen Fall, dass aufgrund einer fehlerhaften Konfiguration das Dashboard von außerhalb des Clusters erreichbar wird zu berücksichtigen, wird empfohlen auch den HTTP respektive HTTPS Verkehr zum und vom Cluster zu überwachen.

#### Spezifische Maßnahmen

Um nicht gewollten Verkehr zu erkennen und zu verhindern, können sogenannte ‘Network-Policies’ in Kubernetes definiert werden. Das verwendete Netzwerk- Plug-in muss die Umsetzung/ Durchsetzung dieser unterstützen. Um ungewollten Netzwerkverkehr zu erkennen, kann der Verkehr zu dem Kubernetes-Dashboard eingeschränkt und Verstöße gegen ‘Network-Policies’ erfasst werden. Viele Netzwerk- Plug-ins, die eine Umsetzung von ‘Network-Policies’ ermöglichen, nutzen iptables für die Realisierung. Wird vom Netzwerk-Plug-in selbst kein Logging bei Verstößen unterstützt, könnten die System-Log-Daten zur Erkennung genutzt werden.

Log-Daten der Kubernetes-API können ebenfalls herangezogen werden. Wird z.B. ein Logging sämtlicher Anfragen an die Kubernetes-API auf dem Log-Level ‘Metadata’ konfiguriert, lässt sich über enthaltene Informationen wie Anfrage-Ursprung und verwendete Autorisierungsdaten erkennen, ob Anfragen von ungewöhnlichen Stellen gesendet oder Autorisierungsdaten genutzt werden, die nicht für den Zeitraum oder die angefragte Ressource intendiert sind (ein Beispiel Log-Eintrag der Kubernetes-API zeigt Listing 3.3).

Siehe auch IZ.2 ‘Spezifischere Logging-/ Monitoring-Maßnahmen’.

```
1 {
2   "kind": "Event",
3   "apiVersion": "audit.k8s.io/v1",
4   "level": "Metadata",
5   "auditID": "c8d543dc-17a1-4e1b-a66d-33d3edb9ed7a",
6   "stage": "ResponseComplete",
7   "requestURI": "/apis/apiregistration.k8s.io/v1?timeout=32s",
8   "verb": "get",
9   "user": {
10    "username": "system:serviceaccount:kube-system:resourcequota-controller",
11    "uid": "47cf8e85-1566-4a3b-8201-61b33e3491cf",
12    "groups": [
13     "system:serviceaccounts",
14     "system:serviceaccounts:kube-system",
15     "system:authenticated"
16   ]
17 },
18   "sourceIPs": [
19    "185.242.x.x"
20   ],
21   "userAgent": "kube-controller-manager/v1.19.11 (linux/amd64)
kubernetes/c6a2f08/system:serviceaccount:kube-system:resourcequota-controller",
22   "responseStatus": {
23     "metadata": {
24     },
25     "code": 200
26   },
27   "requestReceivedTimestamp": "2021-09-25T09:57:31.894544Z",
28   "stageTimestamp": "2021-09-25T09:57:31.894705Z",
29   "annotations": {
30     "authentication.k8s.io/legacy-token": "system:serviceaccount:kube-system:resourcequota-controller",
31     "authorization.k8s.io/decision": "allow",
32     "authorization.k8s.io/reason": "RBAC: allowed by ClusterRoleBinding \"system:discovery\" of ClusterRole \"system:discovery\" to Group \"system:authenticated\""
33   }
34 }
```

Listing 3.3: Beispiel Kubernetes-API-Log (Level: 'Metadata')

### 3.2.2 (AU) Ausführung

#### 3.2.2.1 (AU.1) Exec in Container

Siehe AU.2

#### 3.2.2.2 (AU.2) bash/cmd innerhalb des Containers Bedrohung

Es kann für Entwickler:innen und Administrator:innen hilfreich sein, in einem laufenden Container einer Kubernetes-Umgebung ein Kommandozeilen-Programm zu nutzen, um z. B. Probleme leichter identifizieren oder beheben zu können. Je nach verwendetem Image stehen neben der reinen Kommandozeile noch weitere Tools zur Verfügung. Images wie z. B. das der Linux Distribution Ubuntu kommen mit vielen weiteren vorinstallierten Programmen (ähnlich der normalen Server-Installation). Diese hilfreichen Programme können aber auch im Falle einer Kompromittierung von einem/ einer Angreifer:in ausgenutzt werden. Die Ausführung eines Kommandozeilen- Programms in einem Container, die nicht durch Befugte erfolgt, ist also eine potenzielle Bedrohung und eine Anomalie, die als Indikator auf eine (potenzielle) Kompromittierung hindeuten kann.

#### Allgemeine Maßnahmen

Wie im Grundlagen-Kapitel eingeführt, sind Container-Prozesse separierte reguläre Prozesse auf einem Node. Um Kommandozeilen-Programme innerhalb eines Containers zu entdecken, kann also über den Host eine Kontrolle der ausgeführten Prozesse erfolgen. Hier kann versucht werden auf Charakteristika eines Kommandozeilen- Programms zu achten (z. B. es gibt ein aktives tty-Gerät (Verbindung zum Standard-Input)) oder auch auf Meta-Ebene z. B. die Anzahl an Prozessen zu überwachen.

#### Spezifische Maßnahmen

Die Engine des Falco-Projekts zur Erkennung von Bedrohungen kann genutzt werden, um das Ausführen von Kommandozeilen-Programmen in Containern zu erfassen. Kubernetes selbst erfasst diese Ereignisse nicht standardmäßig.

Ab Kubernetes Version 1.22 gibt es eine Alpha-Funktion, die das Starten sogenannte Debug-Container in einem Pod ermöglicht. Hintergrund ist die steigende Verbreitung von sogenannten ‘distroless Images’ (Images bei denen für die Ausführung der eigentlichen Anwendung nicht benötigte Programme, u.A. auch Kommandozeilen- Programme entfernt wurden, [Pro21]) (vgl. [The21b]). Die Debug-Container sollen es ermöglichen, auch auf Instanzen von ‘distroless Images’ zugreifen zu können, um z. B. Dateien im Container im Rahmen einer Fehlersuche zu sichten. Zu Erkennung dieser Debug-Container wird auf AU.3 verwiesen.

*Diese Maßnahme(n) wird/ werden unter Umsetzung 4.2.1 praktisch evaluiert.*



### 3.2.2.3 (AU.3) Neuer Container

#### Bedrohung

Wenn ein:e Angreifer:in Zugang zu der Kubernetes-API haben sollte, kann es ihr/ ihm möglich sein neue Pods respektive Container zu starten. Wird ein schadhafter Pod/ Container gestartet, kann die/ der Angreifer:in weiteren Schaden anrichten und sich z. B. einen Zugang zum Cluster von außen schaffen. Das Schwierige hier ist, dass Kubernetes-Systeme i. d. R. hochdynamisch sind. Die Anzahl an Pods und Containern kann sich durch z. B. automatischen Skalierungsfunktionen schnell und häufig ändern. Das Starten von neuen Pods ist also nicht grundsätzlich ein Indikator für einen Angriff. Es müssen weitere Aspekte berücksichtigt werden, um legitime von illegitimen neuen Container-Instanzen/Starts unterscheiden zu können.

#### Allgemeine Maßnahmen

Um die Dynamik eines Kubernetes-Systems zu berücksichtigen, wird empfohlen bei der Überwachung der Pods respektive Container in einem Cluster nicht auf Attribute wie die Anzahl der Pods/ Container allein zu vertrauen. Es sollte eine inhaltliche Überprüfung z. B. über das Image und die Namen der Ressourcen erfolgen. Die Anzahl der Pods/ Container kann natürlich trotzdem erfasst werden, denn ein ungewöhnlicher Anstieg dieser Anzahl kann dennoch ein Indikator für eine Kompromittierung sein.

#### Spezifische Maßnahmen

Das Erstellen von Pods respektive Containern in Pods erzeugt Kubernetes-Events und erfolgt durch Anfragen an die Kubernetes-API. Entsprechend sollten die Log-Daten der API ausgewertet und die Kubernetes-Events berücksichtigt werden. Für die Übertragung der Log-Daten könnte Filebeat verwendet werden.

```
Kubernetes-API Audit Logging
NAME: pod | APIGROUP: "" | KIND: Pod | VERBS: [CREATE]
NAME: [deployments,replicasets,statefulsets] | APIGROUP: apps | KIND:
[Deployment,ReplicaSet,StatefulSet] | VERBS: [PATCH,UPDATE]
```

Die Anzahl an laufenden Containern im Cluster kann ebenfalls überwacht werden. Hier gilt es die oben genannte Einschränkung zu berücksichtigen. Eine Überwachung der Anzahl von Containern, in Namespaces (logisch getrennte Bereiche innerhalb des Clusters), die keine automatische Skalierung nutzen, ist ein einfacher und effektiver Indikator. Wird eine automatische Skalierung verwendet, sollten die Anzahl mit der Ressourcennutzung der jeweiligen Anwendungs-Instanzen korreliert werden. Das automatische Skalieren in Kubernetes erfolgt in Zusammenhang mit der Überschreitung von Ressourcen-Nutzungsschwellen. Werden neue Container gestartet ohne, dass diese Schwellen überschritten sind, wird der Auslöser nicht die automatische Skalierung sein und dieser Umstand kann als Indikator für auffälliges Verhalten gewertet werden.

Siehe auch IZ.1 'Spezifischere Logging-/ Monitoring-Maßnahmen'

*Diese Maßnahme(n) wird/ werden unter Umsetzung 4.2.2 praktisch evaluiert.*

### 3.2.2.4 (AU.4) SSH-Server läuft im Container

#### **Bedrohung**

Sollte ein SSH-Dienst in einem Container innerhalb des Clusters ausgeführt werden und dieser von außerhalb des Clusters erreichbar sein, dann könnte ein:e Angreifer:in dies ausnutzen, um von extern Zugriff auf Ressourcen innerhalb des Clusters zu erhalten. Z. B. ein:e Mitarbeiter:in könnte, um ihre/ seine Arbeit zu erleichtern, solch einen Dienst bereitstellen, oder auch in böser Absicht, um externen Zugriff zu ermöglichen.

#### **Allgemeine Maßnahmen**

Um ungewollte Zugriffe von außen zu erkennen, sollte hier eine Überwachung des Netzwerkverkehrs erfolgen. Einige Anwendungen die Netzwerkverkehr loggen und analysieren können, sind in der Lage konkrete Protokolle zu erkennen.

#### **Spezifische Maßnahmen**

Es könnte der Netzwerkverkehr auf Node-Ebene mit z. B. Packetbeat erfasst und analysiert werden (vgl. IZ.2 'Spezifischere Logging- / Monitoring-Maßnahmen'). Ebenso könnte durch 'Network-Policies' Netzwerkverkehr reguliert und Verstöße erfasst werden (vgl. dazu IZ.3 'Spezifischere Logging- / Monitoring-Maßnahmen'). Auf Prozess-Ebene können Prozesse mit typischen Bezeichnungen wie 'sshd' oder ähnliche, die in der auftretenden Form nicht vorgesehen sind, eine Auffälligkeit darstellen.

### 3.2.2.5 (AU.5) Sidecar Injection

#### **Bedrohung**

'Ein Pod kann eine Anwendung kapseln, die aus mehreren eng miteinander gekoppelten Containern besteht, die Ressourcen gemeinsam nutzen müssen. Diese gemeinsam genutzten Container bilden eine einzige zusammenhängende Service-Einheit, z. B. ein Container, der Daten, die in einem gemeinsam genutzten Volume gespeichert sind, für die Öffentlichkeit bereitstellt, während ein separater Sidecar-Container diese Dateien aktualisiert oder auffrischt' (Übersetzt aus [The21f]). Die Bedrohung hier liegt in der unautorisierten bzw. ungewollten Ergänzung eines Pods um einen Container - 'ein Sidecar-Container wird injiziert'. Ein Sidecar-Container ist als Teil des Pods in der Lage z.B. Netzwerkverkehr abzufangen. Entsprechend dem Kubernetes-Netzwerkmodell ist der Verkehr innerhalb eines Pods wie 'localhost'- Verkehr zu klassifizieren. Diese Möglichkeit wird von z.B. Service-Mesh Projekten wie Istio genutzt, um die ausgewählten Pods um einen Proxy zu ergänzen, der z.B. Authentifizierungs-, Autorisierungs-, Routing und Logging-Maßnahmen umsetzt. Ein:e Angreifer:in könnte so entsprechend z.B. Verkehr manipulieren, auslesen oder auf Ressourcen des Pods wie z.B. Speicher zugreifen.

#### **Allgemeine Maßnahmen**

Um Sidedecar-Injektionen erkennen zu können, bietet es sich zum einen an, die Log-Daten der Container-Runtime auszuwerten, zum anderen die Kubernetes-API zu überwachen. Es wird außerdem auf AU.3 verwiesen.

### Spezifische Maßnahmen

Da Sidecar-Container in einem Pod von Kubernetes angesehen werden wie jeder reguläre Container des Pods, können Metainformationen hier hilfreich genutzt werden. So kann z. B. die Anzahl der Container je Pod überwacht werden. Weicht diese Anzahl vom regulären/ vorgesehenen Wert ab, kann dies ein Anlass zu weitergehenden Untersuchungen sein.

Kubernetes-API Audit Logging

NAME: \* | APIGROUP: apps | KIND: \* | VERBS: [PATCH,UPDATE]

NAME: pod | APIGROUP: "" | KIND: Pod | VERBS: [PATCH,UPDATE]

Siehe auch AU.3 'Spezifischere Logging-/ Monitoring-Maßnahmen'.

### 3.2.3 (PS) Persistence

#### 3.2.3.1 (PS.1) Backdoor-Container

##### Bedrohung

Für eine:n Angreifer:in ist es oft erstrebenswert, sich längerfristig Zugang zu einem Cluster zu ermöglichen. Unter der Annahme, dass es einer/ einem Angreifer:in gelungen ist grundsätzlichen Management-Zugriff auf das Cluster zu erlangen, könnte sie/ er 'durch die Verwendung von Kubernetes-Controllern wie DaemonSets oder Deployments [versuchen] [...] sicher[zu]stellen, dass eine konstante Anzahl von [böartigen] Containern auf einem oder allen Nodes des Clusters ausgeführt wird' (übersetzt nach [MW20]). Die/ der Angreifer:in könnte so einen permanenten Zugriff aufbauen und darüber weitere Kompromittierungen erreichen.

##### Allgemeine Maßnahmen

Um Versuche der persistenten Zugangsschaffung erkennen zu können, sind vorrangig die Log-Daten der Kubernetes-API heranzuziehen. Die Daten der Container-Runtime könne hier ebenfalls hilfreich sein.

##### Spezifische Maßnahmen

Anders als bei der reinen Betrachtung der Bedrohung durch einen neuen Pod/ Container (vgl. AU.3), liegt hier der Fokus auf der Überwachung der hoch-leveligen 'Workload-Ressourcen'. 'Deployments', 'DaemonSets', 'StatefulSets', 'ReplicaSets' und 'Jobs' sind sogenannte 'Controller' über die sich bestimmte Zustände von Workloads (Pods) beschreiben und verwalten lassen. I. d. R. werden Workloads im Cluster über diese 'Controller' abgebildet und nicht über einzelne, losgelöste Pod-Spezifikationen. Es sollten also die Kubernetes-Ereignisse und vom System erfassten Daten zu diesen 'Controllern' überwacht werden. Konkreter sollten allgemeine Zustände wie die Anzahl von z.B. Deployments überwacht werden.

Kubernetes-API Audit Logging

NAME: \* | APIGROUP: apps | KIND: \* | VERBS: [CREATE,PATCH,UPDATE]

Ein Deployment kann mehrere Instanzen einer Anwendung haben (mehrere Pods). Das Deployment als übergeordnete Einheit behält seine konstante Anzahl allerdings bei. Die Einschränkungen bei mengenbasierten Kriterien, wie in AU.3 beschreiben, gelten hier also nicht. Diese Eigenschaft gilt auch für die anderen genannten Ressourcen-Typen 'DaemonSets', 'StatefulSets', 'ReplicaSets' und 'Jobs'.

### 3.2.3.2 (PS.2) Schreibbarer Host-Pfad-Mount

#### Bedrohung

Die Bedrohung bei dem 'schreibbarer Host-Pfad-Mount'-Szenario liegt in der direkten Zugriffsmöglichkeit auf den Host. Hier hat ein Pod respektive Container direkten Zugriff auf einen Pfad des Dateisystems des Hosts. Die Trenn-Eigenschaft der Virtualisierung wird damit in gewisser Form, begrenzt auf den Dateisystemzugriff, aufgehoben. Wenn ein:e Angreifer:in in der Lage ist einen neuen Container zu erstellen, dann ist es prinzipiell auch möglich einen 'Host-Pfad-Mount' zu spezifizieren. Weitergehende Persistenz kann ein:e Angreifer:in dann durch Änderungen am Host-System erreichen. Z. B. könnten auf dem Host Konfigurationen verändert werden oder CronJobs angelegt werden (vgl. [MW20]).

#### Allgemeine Maßnahmen

Hier sollten zum einen die Maßnahmen von AU.3 berücksichtigt werden, ergänzt, um den besonderen Fokus auf die Spezifikationen des erstellten Containers. Zum anderen sollten Erkennungsmechanismen eingesetzt werden, die Zugriffe auf primär kritische Pfade (z.B. '/'etc') erkennen können.

#### Spezifische Maßnahmen

Konkret bietet sich hier der Einsatz von Falco an. Wird Falco als Daemonset ausgeführt, gibt es auf jedem Node des Clusters eine Instanz. Falco kann auf diesen Hosts schon in der Standard-Konfiguration Zugriffe auf kritische Pfade und Starts von Containern mit Einbindungen kritischer Verzeichnisse erkennen. Zu berücksichtigen ist hier, dass bei einer fortgeschrittenen Kompromittierung (z. B. Möglichkeit das Daemonset von Falco zu verändern), dieser Erkennungsmechanismus außer Kraft gesetzt werden kann.

*Diese Maßnahme(n) wird/ werden unter Umsetzung 4.2.3 praktisch evaluiert.*

### 3.2.3.3 (PS.3) Kubernetes CronJob

#### Bedrohung

Kubernetes-CronJobs sind sehr ähnlich den CronJob-Funktionalitäten eines klassischen Linux-Systems. Sie bieten die Möglichkeit nach regelmäßigen Zeitintervallen bestimmte Jobs zu starten. Bei den Kubernetes-CronJobs werden allerdings entsprechend keine Skripte oder Programme direkt auf dem Host aufgerufen, sondern Pods gestartet in denen eine Aufgabe ausgeführt wird. Ein:e Angreifer:in könnten diese zeitlich wiederkehrende Ausführung nutzen, 'um die Ausführung von böartigem Code zu planen' ( übersetzt aus [MW20]) und z.B. zu versuchen jeweils einen aktuellen Satz an 'Secrets' (in der Kubernetes-API hinterlegte Geheimnisse wie z.B. Passwörter für Anwendungen in Containern) zu extrahieren.

#### Allgemeine Maßnahmen

Zur Erkennung von ungewollten Kubernetes-CronJobs bietet es sich an, die Kubernetes- Ereignisse rund um diese zu erfassen. Vor allem das Anlegen eines CronJobs ist ein Vorgang, der berücksichtigt werden sollte.

#### Spezifische Maßnahmen

Für jede Ausführung eines Kubernetes-CronJobs wird eine Reihe von Ereignissen von dem Kubernetes-System festgehalten. Werden solche Ereignisse z.B. nicht zu den festgelegten Zeiten

ACS-Veröffentlichung: Abschlussarbeit | Entwurf eines Einsatzkonzepts für Monitoring- und Loggingsysteme zur Angriffsdetektion in Kubernetes-orchestrierten Container-Infrastrukturen

erfasst oder es werden mehr Jobs als vorgesehen ausgeführt, deutete dies auf eine Kompromittierung hin.

Die Anzahl der eingerichteten CronJobs könnte auch überwacht werden und eine Veränderung zu einer Alarmierung führen.

Die Log-Daten der Kubernetes-API können ebenfalls eine hilfreiche Quelle sein. Über diese lässt sich das Anlegen eines neuen sowie das Verändern eines existierenden CronJobs erfassen.

Kubernetes-API Audit Logging  
NAME: cronjobs | APIGROUP: batch | KIND: CronJob | VERBS:  
[CREATE,PATCH,UPDATE]

*Diese Maßnahme(n) wird/ werden unter Umsetzung 4.2.4 praktisch evaluiert.*

### 3.2.3.4 (PS.4) Böswilliger Admission Controller

#### **Bedrohung**

Weizman erläutert die Bedrohung durch einen böswillig eingesetzten Admission Controller, wie folgt:

*„Ein Admission-Controller ist eine Kubernetes-Komponente, die Anfragen an den Kubernetes-API-Server abfängt und möglicherweise modifiziert. Es gibt zwei Arten von Admission-Controllern: validierende und mutierende Controller. Wie der Name andeutet, kann ein mutierender Admission-Controllern die abgefangene Anfrage modifizieren und ihre Eigenschaften ändern. Kubernetes verfügt über einen integrierten generischen Admission-Controllern (MutatingAdmissionWebhook). Das Verhalten dieses Admission-Controllerns wird durch einen Admission- Webhook bestimmt, den der Benutzer im Cluster bereitstellt. Angreifer können solche Webhooks nutzen, um sich im Cluster festzusetzen. So können Angreifer beispielsweise die Pod-Erstellungsvorgänge im Cluster abfangen und modifizieren und ihren bösartigen Container zu jedem erstellten Pod hinzufügen“ (übersetzt aus [WM21]).*

#### **Allgemeine Maßnahmen**

Ein kompromittierter Admission-Controller ist im Betrieb schwer zu erkennen und es sollte daher versucht werden eine Kompromittierung selbst zu detektieren. Es sollte versucht werden, die Erstellung beziehungsweise die Konfiguration eines Admission-Controllers über die Log-Daten der Kubernetes Komponenten (hier primär die Kubernetes-API) zu erkennen und eine entsprechende Alarmierung auszulösen. Fortlaufend sollten Anfragen respektive Verarbeitungsvorgänge durch Admission-Controller mit besonderem Fokus beobachtet werden. Es wird dringlichst empfohlen, einen Einsatz von Admission-Controllern nur mit entsprechenden Authentifizierungsmaßnahmen einzuführen.

#### **Spezifische Maßnahmen**

Da dieser Angriff sehr kritische Folgen haben kann, sollte der Kubernetes-API-Server so konfiguriert werden, dass für sämtliche Admission-Controller eine Authentifizierung erforderlich ist. Hierzu kann dem API-Server eine entsprechende Konfiguration mit dem '-admission-control-config-file' Parameter übergeben werden. Es sollte außerdem unbedingt das Logging für das Erfassen von Anfragen an Admission-Controller aktiviert werden. Das Log-Level (in der Kubernetes Dokumentation 'audit level') sollte mindestens auf 'Metadata' gesetzt werden und entsprechende Alerting-Rules im zentralen Logging-System für Anfrage-Ereignisse an Admission-Controller erstellt werden.

Kubernetes-API Audit Logging

NAME: \* | APIGROUP: admissionregistration.k8s.io | KIND: \* |  
VERBS: [CREATE,PATCH,UPDATE]

### 3.2.4 (PE) Privilegien-Eskalation

#### 3.2.4.1 (PE.1) Privilegierter Container

##### Bedrohung

Standardmäßig werden in Kubernetes Container ohne erweiterte Privilegien ausgeführt. Das heißt ihre Abtrennung (vgl. Grundlagenkapitel Containervirtualisierung) ist vollständig und wie vorgesehen. Es besteht allerdings die Möglichkeit, Container mit erweiterten Privilegien auszuführen. Diese Container erhalten dann Zugriff auf so gut wie alle Funktionalitäten des Hosts bzw. Zugriff zu allen Geräten des Hosts (auch alle Festplatten die selber wieder das Betriebssystem des Hosts beinhalten). Ein:e Angreifer:in kann aus einem privilegierten Container nahezu ungehindert 'ausbrechen' und Schadcode, der in einem manipulierten und privilegierten Container ausgeführt wird, kann entsprechenden Schaden anrichten.

##### Allgemeine Maßnahmen

Es wird daher dringend empfohlen, eine spezifische Überwachung nach dem Kriterium der zusätzlichen Privilegien durchzuführen. Dazu sollten entsprechende Anfragen an die Kubernetes-API überwacht werden, sowie zur Laufzeit Erstellungen von privilegierten Containern nachvollzogen werden.

##### Spezifische Maßnahmen

Bezogen auf die Überwachung an der Kubernetes-API sollten die entsprechenden Rechte zur Erstellung solcher Container eingeschränkt und Anfragen zur Erstellung durch entsprechende Rechte-Inhaber spezifisch erfasst werden.

Kubernetes-API Audit Logging

NAME: \* | APIGROUP: apps | KIND: \* | VERBS: [CREATE,PATCH,UPDATE] NAME:  
pods | APIGROUP: "" | KIND: Pod | VERBS: [CREATE,PATCH,UPDATE]

If '.spec.containers[].securityContext.privileged' is true

Ein Einsatz von Falco zur Erkennung von der Ausführung von privilegierten Container bietet sich zusätzlich an. Falco unterstützt eine Erkennung in der Standard- Konfiguration.

```
1 - rule: Launch Privileged Container
2   desc: Detect the initial process started in a privileged container. Exceptions are made
3   for known trusted images.
4   condition: >
5     container_started and container
6     and container.privileged=true
7     and not falco_privileged_containers
8     and not user_privileged_containers
9   output: Privileged container started (user=%user.name user_loginuid=%user.loginuid
10  command=%proc.cmdline %container.info image=%container.image.repository:%container.image.tag)
11  priority: INFO
12  tags: [container, cis, mitre_privilege_escalation, mitre_lateral_movement]
```

Listing 3.4: Falco-Regel: Privilegierter Container

### 3.2.4.2 (PE.2) Cluster-Admin-Binding

#### Bedrohung

Die Kubernetes Sicherheitsfunktionen umfassen eine rollenbasierte Zugriffskontrolle ('Role-based access control (RBAC)'), um den Zugriff auf Ressourcen/ Aktionen im Cluster zu beschränken. Es gibt unter anderem die vor angelegte Cluster-Rolle 'Cluster-Admin' mit der, wie der Name andeutet, hoch privilegierter Zugriff auf das Cluster erlangt werden kann. Die Bedrohung hier geht nun von solchen Angreifer:innen aus, die sogenannte '(role) bindings' vornehmen können. Hat ein:e Angreifer:in es geschafft die Berechtigung zur Erstellung dieser '(role) bindings' zu erlangen, könnte er/sie eine Bindung an die Cluster-Rolle 'Cluster-Admin' erstellen. Der/ die Angreifer:in hätte so seine/ ihre Rechte im Cluster erweitert und wäre u.U. in der Lage kritische Aktionen im Cluster auszuführen.

#### Allgemeine Maßnahmen

Es wird allgemein empfohlen, die Anfragen an die Kubernetes-API hinsichtlich Anfragen zur Veränderung von RBAC-Regeln oder Rollen-Zugehörigkeiten zu erfassen und gesondert zu betrachten.

#### Spezifische Maßnahmen

In den neueren Versionen von Kubernetes gibt es nach der Dokumentation (vgl. [The21g]) nur in zwei Fällen die Möglichkeit höhere Rechte durch ein Ausnutzen der RBAC zu erreichen:

Zum einen, wenn bereits Zugriff auf eine Rolle existiert, die entsprechende Rechte bzw. Rechte des gleichen Levels hat (u.U. beschränkt z.B. auf einen Namespace). Hier sollten allgemeine Maßnahmen zur Erkennung von Missbrauch getroffen werden. Allgemeines und umfassendes Logging und Alerting schafft hier eine gute Erkennungsmöglichkeit. Zum anderen kann eine Rechteausweitung erfolgen, wenn explizit das Recht zum Verändern einer Rolle (oder Cluster-Rolle) gewährt wurde. Das API-Verb, das hierzu überwacht, werden sollte, ist das 'escalate' Verb.

#### Kubernetes-API Audit Logging

NAME: \* | APIGROUP: rbac.authorization.k8s.io | KIND: \* |  
VERBS: [CREATE,PATCH,UPDATE,[ESCALATE]]



### 3.2.4.3 (PE.3) Host-Pfad-Mount

Diese Bedrohung ist gleichzusetzen mit PS.2. Fokus liegt hier für eine:n Angreifer:in allerdings entsprechend nicht in der Schaffung von Persistenz, sondern in dem potenziellen Zugriff auf Ressourcen des Hosts durch den Mount.

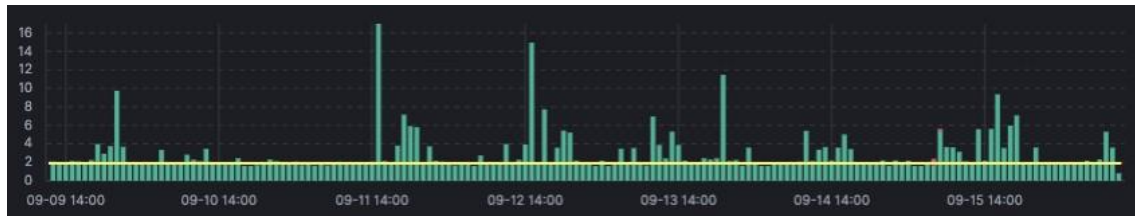


Abbildung 3.2: Kibana: Log-Rate pro Minute

### 3.2.5 (UV) Umgehung der Verteidigung

#### 3.2.5.1 (UV.1) Container-Logs löschen

##### Bedrohung

Ein:e Angreifer:in hat i. d. R. das Bestreben unentdeckt agieren zu können. Eine zentrale Bedrohung für ein System sind also Maßnahmen zur Verschleierung einer Kompromittierung. Das Löschen von Log-Daten und vor allem Container-Log-Daten ist so eine Maßnahme zur Verschleierung. Werden Log-Daten durch eine:n Angreifer:in gelöscht, verringert sich die Nachvollziehbarkeit, die Präzision bei der Erkennung eines Angriffs und Nachweise bzw. die Zuordnung zu der/ dem konkreten Angreifer:in wird erschwert.

##### Allgemeine Maßnahmen

Zur Erkennung dieser Verschleierungsmaßnahmen kann zum einen eine Art Vogelperspektive eingenommen werden. So können Veränderungen in den allgemeinen Strömen von Log-Daten zum zentralen Logging-System einen Verdachtsfall darstellen (der 'einfachste' Fall: es kommen von einem Container keine Log-Daten mehr beim zentralen System an). Auch Abweichungen wie z.B. eine Verringerung der durchschnittlichen Menge an Events in den eintreffenden Log-Daten kann ein Hinweis sein. Darüber hinaus kann versucht werden auf Log-Daten bezogene destruktive Aktionen spezifisch zu erfassen.

##### Spezifische Maßnahmen

Die Abbildung 3.2 zeigt, wie in Kibana (Anwendung des Elastic-Stacks zur Darstellung von u.A. Log-Daten) standardmäßig die minütliche Rate von Log-Einträgen dargestellt wird. Hieran kann man den oben beschriebenen Ansatz nachvollziehen. In der Abbildung gibt es ein erkennbares Grundlevel an Ereignissen (etwa 2 Einträge pro Minute), fällt der Zufluss ab, kann das als Indikator für ein möglichen Angriff angesehen werden und es sollten weitere Untersuchungen folgen.

Neben diesem Ansatz kann versucht werden Löschungen mit Falco zu erkennen. Es ist möglich eine Regel zur Erkennung von Löschvorgängen zu entwerfen (siehe praktische Umsetzung 4.2.5).

*Diese Maßnahme(n) wird/ werden unter Umsetzung 4.2.5 praktisch evaluiert.*



### 3.2.5.2 (UV.2) Kubernetes-Ereignisse löschen

#### Bedrohung

Die Bedrohung hier ist prinzipiell dieselbe wie bei UV.1. Ein:e Angreifer:in versucht hier allerdings nicht einzelne Container-Log-Daten zu vernichten, sondern Kubernetes- Ereignisse zu löschen. Die Kubernetes-Ereignisse sind hochqualitative Informationen über Ereignisse im Cluster (z.B. Container-Erstellung, vgl. AU.3). Ein:e Angreifer:in ist also bestrebt '[...] diese Ereignisse zu löschen (z. B. mit "kubectl delete events -all", um zu verhindern, dass ihre [/seine] Aktivitäten im Cluster entdeckt werden' (übersetzt aus [MW20])).

#### Allgemeine Maßnahmen

Die in UV.1 beschriebenen Maßnahmen können hier ebenfalls angewendet werden. Zusätzlich wäre eine Fokussierung auf Ereignisse die Kubernetes-Ereignisse betreffen sinnvoll.

#### Spezifische Maßnahmen

Die in UV.1 beschriebenen Maßnahmen können hier ebenfalls angewendet werden. Kubernetes-Ereignisse werden in an anderen Stellen produzierten Log-Daten inhaltlich ebenfalls abgebildet. Da es durchaus etwas umständlicher sein kann, Kubernetes- Ereignisse in ein Logging-System zu überführen (es gibt einige Projekte wie den 'kubernetes-event-exporter' [The21k]) sollten vor allem die Log-Daten des Kubernetes- API-Servers überwacht werden. Hier in Hinblick auf Anfragen zur Löschung von Kubernetes-Ereignissen.

Kubernetes-API Audit Logging

NAME: events | APIGROUP: events.k8s.io | KIND: Event |  
VERBS: [PATCH,UPDATE,DELETE,DELETECOLLECTION]

### 3.2.5.3 (UV.3) Ähnlichkeit des Pod-/ Containernamens

#### Bedrohung

Ähnlich zu der genannten Gefahr durch ähnlich benannte Images (s. IZ.1) kann ein:e Angreifer:in durch spezielle Benennung von Instanzen ihres/ seines präparierten Deployments/ Daemonsets/ etc. versuchen ihren/ seinen Angriff zu verschleiern. 'Ein Angreifer könnte zum Beispiel einen böartigen Pod mit dem Namen coredns-zufälliger Suffix erstellen, der zu dem CoreDNS-Deployment verwandt aussähe' (übersetzt aus [MW20])).

#### Allgemeine Maßnahmen

Zur Erkennung eines manipulierten Pods, trotz augenscheinlicher Zugehörigkeit zu einem validen Objekt des Clusters, sollte die in IZ.1 und AU.3 beschriebenen Maßnahmen angewendet werden.

#### Spezifische Maßnahmen

Siehe 'Spezifischere Logging-/ Monitoring-Maßnahmen' IZ.1 und AU.3.

### 3.2.5.4 (UV.4) Verbindung von Proxy-Servern

#### **Bedrohung**

Durch den Einsatz von z.B. Proxy-Servern versucht ein:e Angreifer:in seine ursprüngliche IP-Adresse zu verschleiern und eine entsprechende Reaktion auf die ursprüngliche Quelle sowie eine Rückverfolgung zu erschweren. Vor allem im Nachgang zu einem Angriff sind Informationen wie der Ursprung für die Justierung von präventiven Verteidigungsmaßnahmen sowie für Strafverfolgungsbehörden zur Aufklärung hilfreich.

#### **Allgemeine Maßnahmen**

Eine aktive Rückverfolgung ist aus dem Rahmen von Logging in einem Cluster nicht möglich. Es kann allerdings versucht werden die Spuren eines Angriffs, die direkt erfassbar sind, aufzunehmen. Dazu bietet es sich hier im Netzwerkkontext an, auf ein Logging des Netzwerkverkehrs zurückzugreifen.

#### **Spezifische Maßnahmen**

Da ein:e Angreifer:in in der Regel versuchen wird nicht vorgesehene Zugangswege zu nutzen, bietet es sich an bei Vorhandensein von Firewalls und Netzwerk-Regeln vor allem Verstöße gegen die hier existierenden Regeln zu erfassen und zu untersuchen (s. dazu auch Maßnahmen von IZ.2 und IZ.3).

### 3.2.6 (ZA) Zugang zu Anmelde-informationen

#### 3.2.6.1 (ZA.1) Auslesen K8S-Secrets

##### **Bedrohung**

Secrets in Kubernetes können verwendet werden um z. B. Zugangsdaten, die eine Anwendung für den Zugriff auf eine Datenbank braucht, abzulegen. Die Secrets werden in der Kubernetes-API bzw. dem etcd gespeichert und können in einem Pod referenziert werden. So lassen sich Secrets entweder als Umgebungsvariablen oder als Datei in einen Container einbinden. Diese Secrets können es einer/ einem Angreifer:in unter Umständen ermöglichen weitere Komponenten des Clusters zu kompromittieren. Es gibt drei wesentliche Angriffspfade, die es zu berücksichtigen gilt:

- Sollte es einer/ einem Angreifer:in gelingen Zugriff auf einen Container eines Pods bzw. auf den Service-Account des Pods zu erhalten, können die Secrets auf die dieser Pod Zugriff hat ausgelesen werden. Entweder durch eine Abfrage bei der Kubernetes-API im Namen des Pods unter Verwendung des Service-Accounts oder bei Zugriff auf den Pod, durch Auslesen der Umgebungsvariablen bzw. der als Datei eingebundenen Secrets (vgl. [MW20]).
- Ein anderer Pfad führt über den Host auf dem ein Pod mit eingebundenen Secrets ausgeführt wird.

•

```
1 cat/proc/{containerPID}/environ | tr '\000' '\n'
2
3 SMTP_PASSW=67tHxz5jGi3$!%8dEWRMgU
4 KUBERNETES_PORT=tcp://10.96.0.1:443
5 KUBERNETES_SERVICE_PORT=443
6 NODE_VERSION=14.16.1
7 HOSTNAME=rabbitmq-rest-mail-server-9768756fc-66d98
8 ...
```

### Listing 3.5: Beispiel Kommando & Output

Wie in Listing 3.5 beispielhaft gezeigt, ist es möglich ohne Verwendung von Tools der Container- Runtime die Umgebungsvariablen, aber auch die eingebundenen Dateien auszulesen. Voraussetzung ist, dass der auf dem Host kompromittierte Nutzeraccount Zugriff auf die Dateien hat.

- In der dritten Variante erfolgt das Auslesen bei der Erstellung bzw. der Veränderung von Secrets durch einen schadhafte Admission-Controller (vgl. hierzu PS.4). Sollte ein:e Angreifer:in in der Lage sein einen Admission- Controller erstellen zu können, kann dieser z.B. so konfiguriert werden, dass er bei der 'Überprüfung' jeder Anfrage zur Erstellung oder Bearbeitung von Secrets zwischengeschaltet wird und die Secrets z.B. ausleitet (nach [CG20]).

### Allgemeine Maßnahmen

Viele der bereits in diesem Dokument genannten Maßnahmen und vor allem deren Zusammenwirken, können hier zu einer Erkennung beitragen. Wesentlich ist hier die Überwachung der drei beschriebenen Pfade. Es sollten die Konfiguration und die Zugriffsrechte auf Pods bzw. Container kontrolliert werden, der Zugriff auf die Nodes im Sinne eines Host-Zugriffs und die Erstellung sowie der Einsatz von Admission- Controllern überwacht werden.

### Spezifische Maßnahmen

Es sollten Maßnahmen zur Erkennung von Zugriffen auf Pods bzw. Container ergriffen werden. Siehe hierzu z.B. 'Spezifischere Logging-/ Monitoring-Maßnahmen' von AU.2 und AU.4. Es wird an dieser Stelle außerdem auf die Maßnahmen von PS.4 bezüglich des Admission-Controllers verwiesen. Darüber hinaus können die Log-Daten der Kubernetes-API in Bezug auf das Abfragen von Secrets überwacht werden.

Kubernetes-API Audit Logging  
NAME: secrets | APIGROUP: "" | KIND: Secret |  
VERBS: [GET,LIST,WATCH]

### 3.2.6.2 (ZA.2) Zugang zum Container-Service-Account

#### Bedrohung

In Kubernetes werden im Rahmen der Role-based-authentication (RBAC) sogenannte Service-Accounts verwendet, um einen Dienst (z.B. eine Anwendung bzw. einen Pod) identifizieren zu können. Der Service-Account ist dabei ein Token mit dem sich der Dienst gegenüber der Kubernetes-API identifizieren kann. Standardmäßig wird jedem Pod der Standard-Service-Account (wenn nicht anders konfiguriert) des Namespaces in dem er sich befindet zur Verfügung gestellt. Dies erfolgt durch Bereitstellung per Datei-Mount (Pfad in jedem Container: `/var/run/secrets/kubernetes.io/serviceaccount/token`). Sollte die Authentifizierung über RBAC nicht aktiviert sein, dann hat jeder Service-Account unbegrenzte Rechte. Bei aktivierter RBAC bestimmten ClusterRoleBindings bzw. RoleBindings die genauen Berechtigungen des Service-Accounts. Sollte es einer/ einem Angreifer:in gelingen Zugriff auf einen Container zu erlangen, besteht entsprechend Zugriff auf den eingebundenen Service-Account. Ein:e Angreifer:in kann mit diesem dann authentifizierte Anfragen an die Kubernetes-API senden. Bei deaktivierter RBAC besteht vollständiger Zugriff auf das Cluster. Bei aktivierter RBAC sind die Möglichkeiten der/ des Angreiferin/ Angreifers beschränkt auf die zugewiesenen Rechte des Service-Accounts. ([MW20], [The21h])

#### Allgemeine Maßnahmen

Allgemein sei zunächst angemerkt, dass es dringend zu empfehlen ist, die RBAC zu aktivieren. Dies ist über den API-Server-Parameter `--authorization-mode=[...],RBAC` zu realisieren. RBAC ist bei neueren Kubernetes-Versionen standardmäßig aktiviert. Eine entsprechende Konfiguration mit dem Ziel nur nötige Rechte an einen Service-Account zu vergeben ist ebenfalls zu empfehlen. Wenn keine eigenen Konfigurationen vorgenommen wurden, hat ein Service-Account bei aktivierter RBAC keine Rechte. Die Erkennung des Missbrauchs eines Service-Accounts kann im Rahmen von Anfragen, die sich nicht im Rahmen der Rechte des Service-Accounts bewegen, durch Überwachung von abgelehnten Anfragen an die Kubernetes-API erfolgen. Bei Missbrauch im Rahmen der Rechte des Service-Accounts wird die Erkennung entsprechend erschwert. Hier können andere Indikatoren zur grundsätzlichen Feststellung einer Kompromittierung herangezogen werden oder es kann versucht werden auffällige Verhaltensweisen zu erkennen. Das heißt Verhalten, dass auf einen Service-Account zurückzuführen ist, wird mit bisherigem Verhalten verglichen und eine Abweichung wird als Auffälligkeit gemeldet. Diese verhaltensbasierten Verfahren werden in der Praxis häufig mithilfe von Machine-Learning realisiert, stellen aber oft kostenpflichtige Zusatzfunktionen dar und werden daher hier nicht weiter behandelt.

#### Spezifische Maßnahmen

Um eine Erfassung konkret zu ermöglichen, ist vor allem die Konfiguration der Kubernetes-API in Bezug auf Logging (in der Kubernetes Dokumentation 'Auditing') von Relevanz. Ein Minimum an Log-Daten kann z.B. durch diese Konfiguration in Listing 3.6 erreicht werden. Bei dem angegebenen Beispiel werden sämtliche Anfragen an die Kubernetes-API erfasst und ihre Metadaten in Log-Einträgen übernommen. Fehlgeschlagene Anfragen lassen sich hierdurch erkennen und es kann ihnen nachgegangen werden. Vor allem wenn die Rate entsprechend auffällig sein sollte.

```
1 # Log all requests at the Metadata level.  
2 apiVersion: audit.k8s.io/v1  
3 kind: Policy  
4 rules:  
5 - level: Metadata
```

Listing 3.6: Log-Konfiguration Kubernetes-API ([The21a])

### 3.2.6.3 (ZA.3) Böswilliger Admission controller

Siehe 'Bedrohung', 3. Angriffspfad von ZA.1 und allgemein PS.4.

## 3.2.7 (EK) Erkundung

Für eine:n Angreifer:in sind (weitergehende) Informationen über ein Cluster und seine Komponenten hilfreich, um sich besser im Cluster ausbreiten zu können und weitere Komponenten kompromittieren zu können. Dieser Abschnitt betrachtet Fälle die in diese Erkundungs-Maßnahmen einzuordnen sind.

### 3.2.7.1 (EK.1) Zugriff auf den Kubernetes-API-Server

#### **Bedrohung**

Im Sinne der Informationsgewinnung ist der Kubernetes-API-Server eine primäre Quelle. Sollte es einer/ einem Angreifer:in möglich sein, den Kubernetes-API- Server anzusprechen und erfolgreich abzufragen, hat die/ der Angreifer:in u.U. die Möglichkeit detaillierte und kritische Informationen über das Cluster aus primärer Quelle zu erlangen. Ist es einer/ einem Angreifer:in gelungen das Cluster bereits zu kompromittieren und es besteht z.B. Zugriff auf/ von einem Container aus, dann kann die/ der Angreifer:in u.U. durch Nutzung des Container-Service-Accounts autorisiert Informationen abfragen (vgl. ZA.2). Sollte ein Cluster gegenüber externen Quellen nicht ausreichend gesichert sein, kann ein:e Angreifer:in versuchen Anfragen an die Kubernetes-API zu senden. So lassen sich allgemeine Informationen sammeln, wie die verwendete Version des Clusters.

#### **Allgemeine Maßnahmen**

Anfragen an die Kubernetes-API sollten also grundsätzlich überwacht werden, um z.B. auf Basis einer veränderten Rate von Anfragen weitere Untersuchungen vorzunehmen. Auch könnten präzisere Informationen wie der Ursprung der Anfrage (z.B. von einem Container aus, der normalerweise keine Anfragen an die Kubernetes-API sendet) als Indikator für eine mögliche Kompromittierung dienen. Um Anfragen von außen zu erkennen bietet sich der Einsatz von Regulationsmechanismen auf Netzwerkebene an (z.B. Firewall/ Netzwerk-Regeln). Verstöße können hier ebenfalls erfasst werden und als Hinweis dienen.

### Spezifische Maßnahmen

Siehe allgemein ZA.2 und ‘Spezifischere Logging-/ Monitoring-Maßnahmen’ von:

- IZ.2
- IZ.3
- AU.2

#### 3.2.7.2 (EK.2) Zugang zur Kubelet-API

##### Bedrohung

Kubelet ist ein Programm, welches auf jedem Node des Clusters ausgeführt wird. Es ist die lokale Repräsentation von Kubernetes und verwaltet alle für den Node spezifischen Ereignisse (z.B. korrekte Ausführung eines dem Node zugewiesenen Pods). Kubelet hat eine eigene API-Schnittstelle, die z.B. Informationen über den Node (z.B. CPU Auslastung) oder die Pods auf dem Node bereitstellt. In älteren Kubernetes Versionen konnten diese Informationen ohne Autorisierung abgefragt werden. Inzwischen ist eine Autorisierung standardmäßig notwendig.

##### Allgemeine Maßnahmen

Da die Kubelet-API an den Node gebunden (‘https://localhost|NodeIP:10255/’) ist kann eine Überwachung auf Node-Ebene durchgeführt werden. Wird der Netzwerkverkehr auf dem Node überwacht, dann wird damit der Fall, dass Anfragen von außen gesendet werden, aber auch der Fall, dass Anfragen von innerhalb des Clusters gesendet werden abgedeckt. Indikatoren, die ein auffälliges Verhalten andeuten bzw. auf einen Angriff hindeuten können, sind zum einen der Ursprung der Anfragen (vor allem von außen) und die Rate/ der Zeitpunkt der Abfragen.

### Spezifische Maßnahmen

Siehe ‘Spezifischere Logging-/ Monitoring-Maßnahmen’ von:

- IZ.2
- IZ.3
- AU.2

#### 3.2.7.3 (EK.3) Netzwerk-Mapping

##### Bedrohung

Im Rahmen der Erkundung eines Ziel-Systems ist es für eine:n Angreifer:in aufschlussreich, Aufklärung der vorhandenen Netzwerkstrukturen zu betreiben. Informationen über erreichbare Dienste und die Netzwerktopologie können einer/ einem Angreifer:in helfen das Ziel-System weiter zu kompromittieren. Sollte ein:e Angreifer:in Zugriff auf einen Node des Clusters oder einen Container des Clusters erlangt haben, wird sie/ er u.U. sogenannte Netzwerkscans durchführen und Dienste auf konkrete Schwachstellen hin testen. In der Standardkonfiguration von Kubernetes gibt es keine Einschränkungen der Kommunikation von Containern bzw. Pods (vgl. Grundlagen: Kubernetes Netzwerk-Komponenten und [MW20]).

##### Allgemeine Maßnahmen

Um Netzwerk bezogene Aufklärungsmaßnahmen einer/ eines Angreiferin/ Angreifers zu erkennen sollte der Netzwerkverkehr überwacht und analysiert werden. Auffälligkeiten wie ungewöhnlicher

[ACS-Veröffentlichung: Abschlussarbeit](#) | Entwurf eines Einsatzkonzepts für Monitoring- und Loggingsysteme zur Angriffsdetektion in Kubernetes-orchestrierten Container-Infrastrukturen

Netzwerkverkehr (Kommunikation von/ zu nicht vorgesehenen Quellen/ Zielen) oder eine veränderte Menge von Anfragen an einen Dienst können Hinweise auf eine Kompromittierung sein. Effektiv kann auch die Erkennung von Verstößen gegen implementierte Firewall- und Kubernetes-Netzwerk-Regeln sein. Sollte ein:e Angreifer:in bei Ihren Aufklärungsmaßnahmen gegen solche Regeln verstoßen, schützt die Regel zum einen vor den Maßnahmen durch die/den Angreifer:in, aber auch die Erkennung wird erheblich erleichtert.

### **Spezifische Maßnahmen**

Siehe 'Spezifischere Logging-/ Monitoring-Maßnahmen' von:

- IZ.2
- IZ.3
- AU.2

*Diese Maßnahme(n) wird/ werden unter Umsetzung 4.2.6 praktisch evaluiert.*

#### **3.2.7.4 (EK.4) Zugang zum Kubernetes Dashboard**

Siehe Bedrohung IZ.3.

#### **3.2.8 (LM) Lateral Movement**

##### **3.2.8.1 (LM.1) Container-Service-Account**

Siehe Bedrohung ZA.2.

##### **3.2.8.2 (LM.2) Cluster-internes Networking**

Ein:e Angreifer:in kann die flache und im Standardfall uneingeschränkte Netzwerkstruktur von Kubernetes nutzen, um sich im Cluster weiter auszubreiten. (vgl. Grundlagen: Kubernetes Netzwerk-Komponenten und [MW20]).

Diese Bedrohung ist vergleichbar mit: EK.3.

Für allgemeine und spezifische Maßnahmen siehe:

- EK.3
- IZ.2
- IZ.3
- AU.2

*Diese Maßnahme(n) wird/ werden unter Umsetzung 4.2.6 praktisch evaluiert.*

##### **3.2.8.3 (LM.3) Schreibbare Volumemounts auf dem Host**

Siehe Bedrohung PS.2.

##### **3.2.8.4 (LM.4) Zugang zum Kubernetes Dashboard**

Siehe Bedrohung IZ.3.

### 3.2.8.5 (LM.5) CoreDNS-Manipulation

#### Bedrohung

Kubernetes interner DNS-Dienst (CoreDNS) wird von den Containern im Cluster standardmäßig zur Auflösung einer Domain angesprochen. Sollte es einer/ einem Angreifer:in gelingen die Konfiguration des internen DNS Dienstes zu manipulieren, kann damit i. d. R. ein Großteil der Netzwerkkommunikation kompromittiert werden. Die konkrete Bedrohung liegt hier z.B. in der Umleitung von Anfragen an ein Ziel, welches unter der Kontrolle der/ des Angreiferin/ Angreifers steht. Möglich wäre so eine Manipulation durch Veränderung der CoreDNS-Konfiguration welche in einer ConfigMap gespeichert ist. Ebenfalls wäre eine Manipulation über die Veränderung der DNS-Konfiguration des Hosts ('resolv.conf'), welche CoreDNS nutzt, um Cluster externe Adressen auflösen zu können, möglich.

#### Allgemeine Maßnahmen

Da diese Bedrohung Netzwerk basiert ist, können Maßnahmen zur Eindämmung und Erkennung wie z.B. der Einsatz von Firewall- und Kubernetes-Netzwerk-Regeln wiederverwendet werden. Siehe dazu:

- IZ.2
- IZ.3
- AU.2

#### Spezifische Maßnahmen

Um eine Veränderung der ConfigMap des CoreDNS-Dienstes zu erfassen lässt sich auf Daten der Kubernetes-API zurückgreifen:

Kubernetes-API Audit Logging

NAME: configmaps | APIGROUP: " " | KIND: ConfigMap | VERBS: [PATCH,UPDATE]

Siehe 'Spezifischere Logging-/ Monitoring-Maßnahmen' von:

- IZ.2
- IZ.3
- AU.2

### 3.2.8.6 (LM.6) ARP-Spoofing

#### Bedrohung

Kubernetes Netzwerk-Plugins realisieren das von Kubernetes geforderte Overlay- Netzwerk i. d. R. über eine Struktur, bei der eine zentrale Bridge an das eigentliche Netzwerk-Interface des Hosts angeschlossen ist. An diese Bridge sind dann die virtuellen Netzwerkschnittstellen der Pods angeschlossen (siehe auch Listing 3.7). Viele Netzwerk- Plugins nutzen das 'Address Resolution Protocol' (ARP, [PI82]), um die Ziel-MAC- Adresse eines Netzwerk-Pakets aufzulösen. Dies geschieht an der zentralen Bridge. Wie bei LM.5 eingeführt, nutzen die Pods in Kubernetes standardmäßig den Cluster internen DNS-Dienst, um Domain-Namen aufzulösen. In jedem Pod wird die IP- Adresse '10.96.0.10' als DNS-Server konfiguriert. Diese IP ist eine virtuelle Adresse die per 'Destination Network Address Translation'



(DNAT) auf die tatsächliche Adresse des CoreDNS umgesetzt wird. Der kube-proxy (siehe auch Grundlagen) ist für das Setzen der entsprechenden iptables Regeln verantwortlich. Die Bedrohung liegt hier nun darin, dass ein:e Angreifer:in aus einem Pod heraus die Adress-Umsetzung bzw. die Auflösung der CoreDNS-Adresse manipulieren kann. Standardmäßig erhalten Pods in Kubernetes die 'NET\_RAW'-Fähigkeit. Dies ist intendiert, um z.B. ICMP-Pakete ('ping') an andere Pods senden zu können. Dadurch kann ein:e Angreifer:in allerdings 'rohe' Netzwerkpakete erstellen (inklusive ARP-Paketen). Die/ Der Angreifer:in könnte nun vorgeben, dass z.B. ihr/sein Pod der cluster-interne DNS-Dienst sei indem sie/ er die Adressauflösung hin zu ihr/ihm als Ziel manipuliert (ARP-Spoofing). ([SA19])

```
1 vxlan.calico: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1440
2     inet 10.244.71.192 netmask 255.255.255.255 broadcast 0.0.0.0
3     inet6 fe80::647b:95ff:fe5f:f3b prefixlen 64 scopeid 0x20<link>
4     ether 66:7b:95:5f:0f:3b txqueuelen 0 (Ethernet)
5     ...
```

Listing 3.7: Core Bridge und veth-Interface des Calico CNI

```
1 root@containerInPod:~# cat /etc/resolv.conf
2 nameserver 10.96.0.10
3 search namespace.svc.cluster.local svc.cluster.local cluster.local
4 options ndots:5
```

Listing 3.8: DNS-Konfiguration eines Containers in Kubernetes

## Allgemeine Maßnahmen

Um einen realen Fall dieser Bedrohung erkennen zu können, sollten auf Ebene des Hosts und des Kubernetes-Netzwerks grundlegende Maßnahmen ergriffen werden. Die Kommunikation sollte auf das Notwendigste beschränkt werden. Ebenso sollten Rechte der Pods bzw. Container auf das Notwendigste reduziert werden.

## Spezifische Maßnahmen

Konkret heißt das, dass z.B. die 'NET\_RAW'-Fähigkeit nur in solchen Fällen gewährt werden sollte in denen es zwingend erforderlich ist. In den meisten Anwendungsfällen wird diese Fähigkeit nicht benötigt. Dienste wie etwa VPN-Dienste oder Komponenten des Netzwerk-Plug-ins, welche das Kubernetes-Overlay-Netzwerk realisieren sind Repräsentanten für die kleine Klasse an Diensten die diese Fähigkeit wirklich benötigen. Listing 3.9 zeigt wie die 'NET\_RAW'-Fähigkeit entzogen werden kann.

```
1 ...
2 containers:
3   - name: abc
4     image: xyz
5     securityContext:
6       capabilities:
7         drop:
8           - NET_RAW
```

Listing 3.9: Entziehen der 'NET\_RAW'-Fähigkeit

## 4 Praktische Umsetzung

In diesem Kapitel werden die Ergebnisse der theoretischen Untersuchung in Form einer praktischen Umsetzung evaluiert und überprüft. Im Rahmen dieser Arbeit können nicht alle im theoretischen Teil behandelten Angriffe und Taktiken untersucht werden. Es wird sich in diesem Teil auf die folgenden Angriffe beschränkt:

- bash/cmd innerhalb des Containers (AU.2)
- Neuer Container (AU.3)
- Schreibbarer Host-Pfad-Mount (PS.2)
- Kubernetes CronJob (PS.3)
- Container-Logs löschen (UV.1)
- Netzwerk-Mapping bzw. Cluster-internes Networking (EK.3 bzw. LM.2)

Diese Auswahl wurde unter der Berücksichtigung der repräsentativen Stellung der Angriffe (möglichst Angriffe aus verschiedenen Taktik-Gruppen), der Umsetzung (möglichst verschiedene Maßnahmen zur Erkennung) und dem Aufwand der Umsetzung getroffen. Die oben aufgeführten Angriffe bilden einen guten Querschnitt durch die Gesamtheit der untersuchten Angriffe, sind aber auch in für diese Arbeit angemessenem Rahmen praktisch umsetzbar.

### 4.1 Infrastruktur & Aufbau des Experiments

Als grundlegende Infrastruktur wurde für dieses Experiment ein Kubernetes-Cluster (Version 1.19.11) mit zwei Nodes aufgesetzt. Der Control-Plane-Node wurde für Workloads freigegeben, um ausreichend Ressourcen für alle benötigten Dienste bereitstellen zu können. Das Overlay-Netzwerk ist ein VXLAN-basiertes und von dem Plug-in Calico realisiertes virtuelles Netzwerk zwischen den

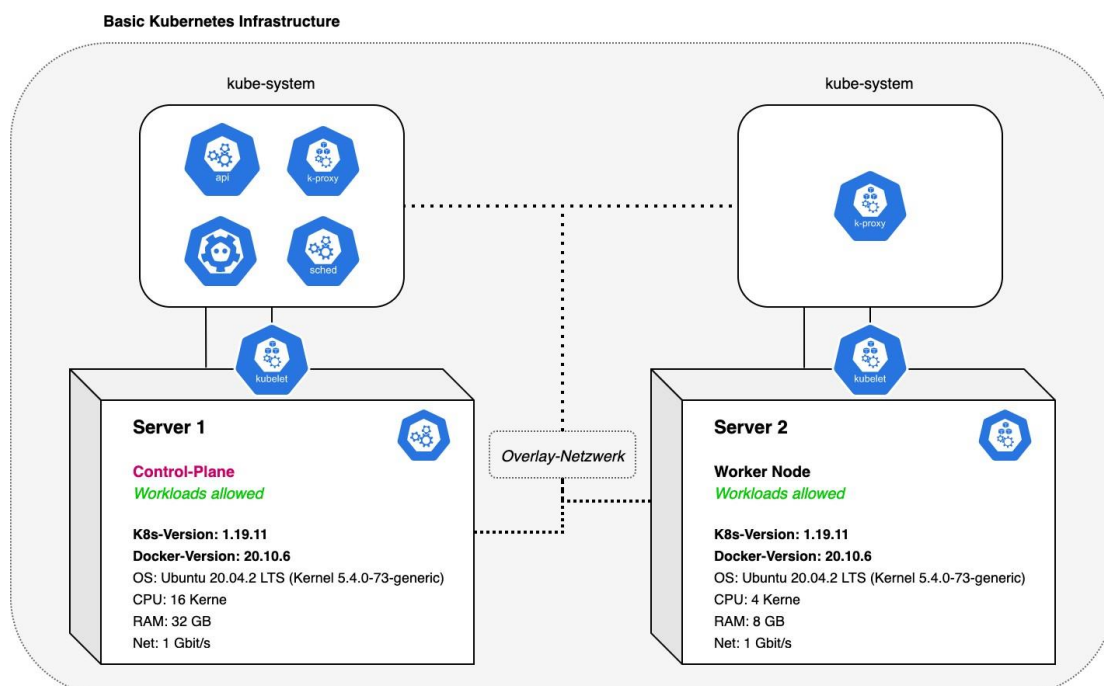


Abbildung 4.1: Grundlegende Infrastruktur

Nodes. Als Container- Runtime wird Docker (Version 20.10.6) verwendet. Die Abbildung 4.1 veranschaulicht die grundlegende Struktur. Auf dem Control-Plane-Node werden entsprechend die wesentlichen Kubernetes System-Komponenten ausgeführt. In diesem Kubernetes-Cluster wurde ein zentrales Logging-System auf Basis von Diensten des Elastic-Stacks installiert. Außerdem wurde Falco sowie eine UI Komponente und ein Event-Shipper (Weiterleitung von Falco-Events an Elasticsearch-Datenbank) installiert. Dieses System bzw. diese Dienste sollen es ermöglichen, die ausgewählten Angriffe zu detektieren.

#### 4.1.1 Elastic-Stack basiertes zentrales Logging

Zentraler Speicherpunkt für sämtliche Log-Daten in dem aufgebauten System ist die Elasticsearch-Datenbank. Es wurde eine zentralisierte Architektur gewählt. Ausschlaggebend für diese Entscheidung sind die überwiegenden Vorteile wie z. B. ein einzelner Zugriffspunkt trotz vieler verteilter Quellen und die damit verbundene Möglichkeit zur Korrelation. Diese dokumenten-basierte Datenbank arbeitet intern mit separierten Indizes (vergleichbar/ vorstellbar wie verschiedene Datenbanken eines klassischen Datenbanksystems). Es wurde für jede datenzuführende Komponente ein Index angelegt:

- **[Falco] falco-\***: Daten die von der Falco Threat-Detection-Engine erzeugt und über den Dienst Falco-Sidekick in die Elasticsearch-Datenbank übertragen wurden (siehe auch Falco und Zusatz-Dienste). Enthält Daten zu den von Falco erfassten Ereignissen.
- **[Metricbeat] metricbeat-\***: In diesem Index werden Daten der Metricbeat Instanzen gespeichert. Diese Daten enthalten Angaben zu der physischen Ressourcenauslastung der Nodes, wie z.B. CPU oder Arbeitsspeicher Belegung.
- **[Filebeat] filebeat-\***: Filebeat ist ein sogenannter Log-Shipper. Das heißt, Filebeat überträgt von anderen Systemen erzeugte Log-Daten in die Elasticsearch-Datenbank. In dieser Infrastruktur wurde die standardmäßige Übertragung von Log-Daten aller Container deaktiviert (Anwendungsspezifische Log-Daten werden in dieser Arbeit nicht berücksichtigt). Filebeat wurde so konfiguriert, dass wesentliche systemspezifische Log-Daten (z.B. der Kubernetes- API oder des Host-Systems) übertragen werden.
- **[Packetbeat] packetbeat-\***: Um vor allem den Cluster-internen Netzwerkverkehr überwachen zu können und spezifische Protokolle (hier primär DNS und HTTP) sowie grundsätzlich Meta-Daten wie Quelle und Ziel analysieren zu können, wird Packetbeat verwendet. Der angelegte Index enthält entsprechende Netzwerkdaten (aller Pods und der Nodes) und Auswertungen.

Falco sowie die verwendeten Beats des Elastic-Stacks werden als Daemonset im Cluster ausgeführt. Es wird von diesen Anwendungen auf jedem Host des Clusters eine Instanz ausgeführt. Es wurde im Sinne der Forensik und Qualität der Log-Daten (Integrität und Vertraulichkeit) möglichst durchgängig eine TLS-Verschlüsselung der Übertragungswege eingesetzt.

Zugänglich gemacht werden die erfassten Daten durch die Frontend-Komponente des Elastic-Stacks (Kibana). Hier werden die Daten visualisiert und können von einer/ einem Verantwortlichen abgerufen werden.

#### 4.1.2 Falco und Zusatz-Dienste

Zur Erkennung von präzisen Spuren und Indikatoren (wie z. B. Zugriffe auf kritische Pfade des Hosts oder der Ausführung von Kommandozeilen-Programmen in einem Container) wird Falco eingesetzt. Die Funktionsweise von Falco wurde im Grundlagenkapitel dargestellt (vgl. Grundlagen Falco). Die eingesetzten Regeln, die Falco zur Überprüfung nutzt, werden hier den Instanzen über eine

ConfigMap zur Verfügung gestellt. Für den Fall einer Alarmierung wurde Falco so konfiguriert, dass ein externer Webhook des Kommunikationsdienstes Slack aufgerufen und in eine dort angelegte Gruppe eine entsprechende Nachricht abgesetzt wird. Nach dem gleichen Prinzip (HTTP-basiert) werden die Ereignisse an Falco-Sidekick übertragen. Dieser Dienst speichert die erfassten Ereignisse zwischen und überträgt die Daten an die Elasticsearch-Datenbank. Der Dienst Falco-Sidekick-UI greift auf die zwischengespeicherten Daten von Falco-Sidekick zu und ermöglicht eine direkte Ansicht der von Falco erfassten Daten über eine Weboberfläche.

### 4.1.3 Infrastruktur-Übersicht

Die Abbildung 4.2 zeigt übersichtsartig die aufgebaute Struktur. Auf Basis dieser Struktur wird im Weiteren aufgebaut und ihre Eigenschaften und Komponenten vorausgesetzt.

## 4.2 Durchführung

### 4.2.1 bash/cmd innerhalb des Containers (AU.2)

#### Erkennungsansatz & Realisierung

Es werden zwei Ansätze verfolgt, um diese Bedrohung (vgl. 'bash/cmd innerhalb des Containers (AU.2)') zu erkennen:

**[Falco] 'Live'-Erkennung über Syscalls und Falco-Engine:** Die im Folgenden angegebene Falco-Regel ist eine Standard-Regel des Projekts und ermöglicht die Erfassung von Ausführungen von Kommandozeilen-Programmen.

```
1 spawned_process and container
2 and shell_procs and proc.tty != 0
3 and container_entrypoint
4 and not user_expected_terminal_shell_in_container_conditions
```

Listing 4.1: Falco-Regel: Kommandozeilenprogramm - Bedingungen

Die Bedingung (sh. Listing 4.1), die zum Auslösen einer Meldung erfüllt werden muss, setzt sich aus folgenden Teilen zusammen: Zum einen muss ein neuer Prozess gestartet worden sein, welcher einem Container zugehörig ist und nicht der eigentliche definierte Entrypoint (Spezifizierung des Programms und seiner Attribute, welches beim Start des Containers aufgerufen werden soll) ist. Zusätzlich muss der Prozessname einem Namen eines Kommandozeilenprogramms (z. B. 'bash' oder 'zsh') entsprechen und es muss ein tty-Gerät gebunden sein. Abschließend wird geprüft, ob der Start durch eine:n Nutzer:in erwartet wird. Wird ein Kommandozeilenprogramm in einem Container gestartet, erfasst Falco dies. Die oben vorgestellte Regel löst eine Meldung aus, auf welche reagiert werden kann.

```
1 - macro: user_expected_terminal_shell_in_container_conditions
2     condition: (never_true)
3
4 - rule: Terminal shell in container
5     desc: A shell was used as the entrypoint/exec point into a container with an attached terminal.
6     condition: >
7         spawned_process and container
8         and shell_procs and proc.tty != 0
9         and container_entrypoint
10        and not user_expected_terminal_shell_in_container_conditions
11    output: >
12        A shell was spawned in a container with an attached terminal (user=%user.name
13        user_loginuid=%user.loginuid %container.info
14        shell=%proc.name parent=%proc.pname cmdline=%proc.cmdline terminal=%proc.tty
15        container_id=%container.id image=%container.image.repository)
16    priority: NOTICE
17    tags: [container, shell, mitre_execution]
```

Listing 4.2: Falco-Regel: Kommandozeilenprogramm

## Visualisierung/ Alarmierung

Eine Visualisierung bzw. Alarmierung erfolgt hier zum einen über den Aufruf eines Webhook Endpunktes des Kommunikationsanbieters Slack, zum anderen durch Senden der Ereignisse an Falco-Sidekick und die dortige Übermittlung an die Elasticsearch- Datenbank. Das Listing 4.3 sowie 4.4 zeigen die dafür notwendige Konfiguration.

```
1 programOutput:
2   enabled: true
3   program: |
4     jq `if .priority == "Emergency" or .priority == "Critical" or .priority == "Error" then
5       { attachments: [{ text: .output, color: "danger" }] }
6     elif .priority == "Warning" or .priority == "Notice" then
7       { attachments: [{ text: .output, color: "warning" }] }
8     elif .priority == "Informational" then
9       { attachments: [{ text: .output, color: "good" }] }
10    else
11      { attachments: [{ text: .output }] }
12    end' | curl -d @- -X POST https://hooks.slack.com/services/xyz/abc/xyz123
13
14 httpOutput:
15   enabled: true
16   url: "http://falcosidekick:2801/"
```

Listing 4.3: Falco-Konfiguration: Ausgabe

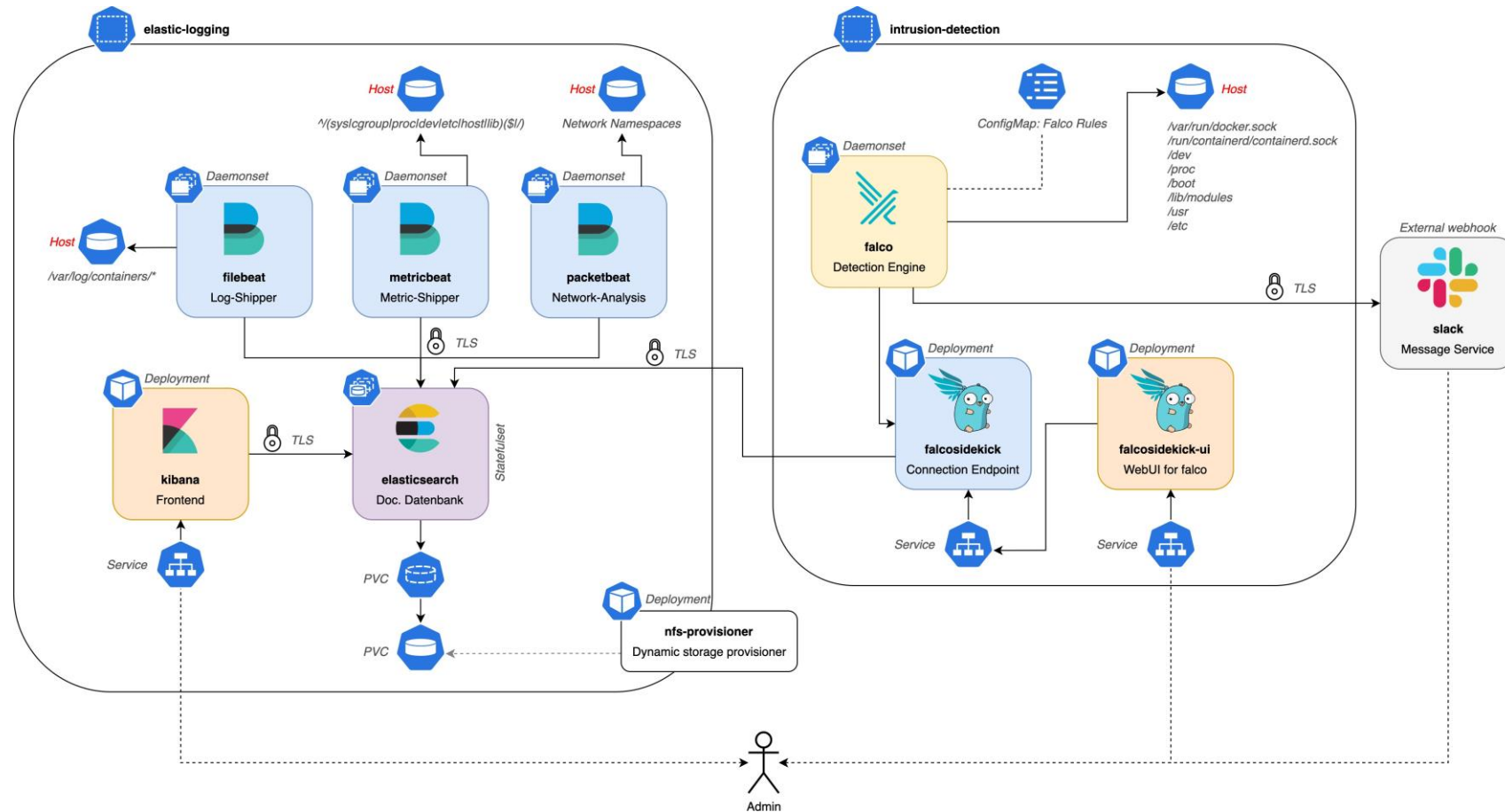


Abbildung 4.2: Übersicht der Logging Infrastruktur

```
1 elasticsearch:
2   hostport: "https://elasticsearch-master.elastic-logging.svc.cluster.local:9200"
3   index: "falco"
4   type: "event"
5   minimumpriority: ""
6   mutualtls: false
7   checkcert: true
8   username: "falco"
9   password: "superSecret"
```

Listing 4.4: Falco-Sidekick-Konfiguration: Ausgabe

Durch den Aufruf von 'kubectl exec' wurde ein Kommandozeilenprogramm in einem Container des Clusters gestartet. Eine Alarmierung erfolgte durch eine Slack-Nachricht (s. Abbildung 4.3). Außerdem lässt sich, wenn in Kibana eine Visualisierung für die Meldungen von Falco erstellt wird, bei der die Anzahl an Ereignissen je Ereignistyp in Form eines Balkendiagramms dargestellt wird, erkennen, dass dieses Ereignis aufgetreten ist. Abbildung 4.4 zeigt ein solches Balkendiagramm.

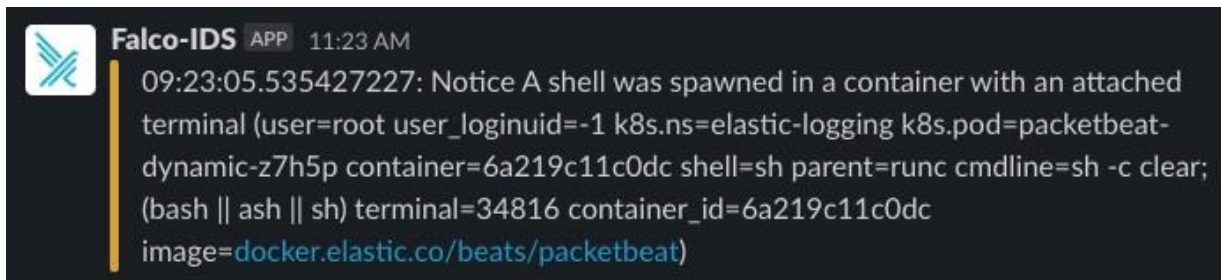


Abbildung 4.3: Slack-Nachricht: Falco Ereignis - Start Kommandozeilenprogramm



Abbildung 4.4: Kibana-Balkendiagramm: Falco Ereignisse



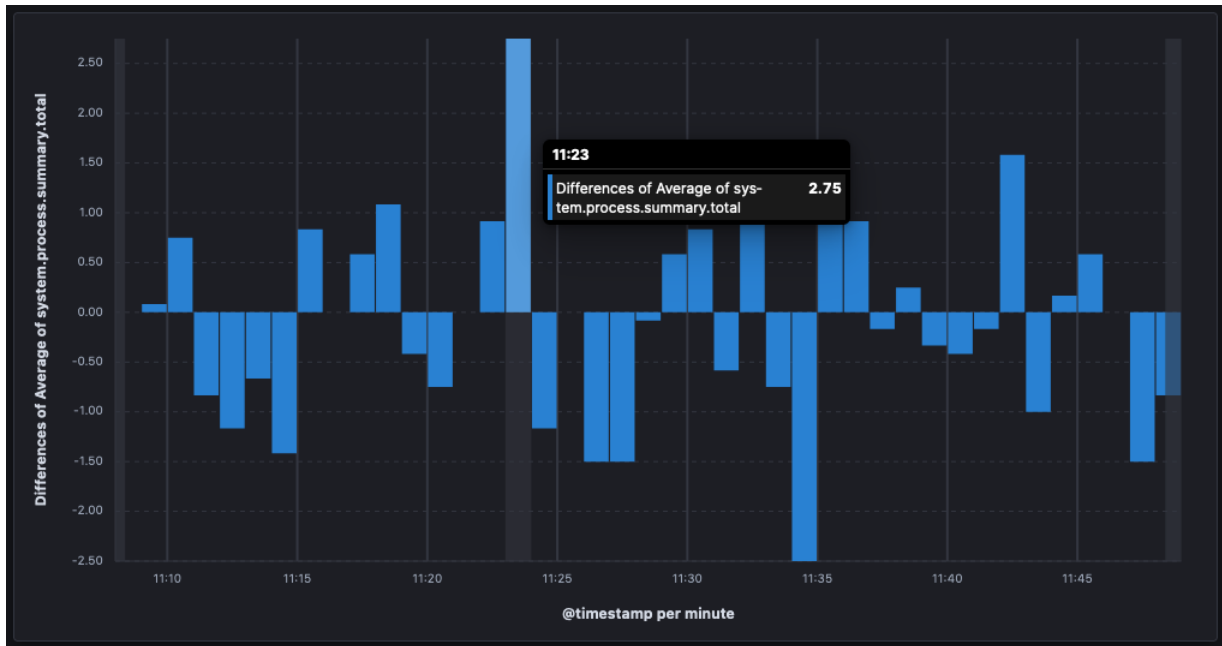


Abbildung 4.5: Kibana-Balkendigramm: Abweichung von durchschnittlicher Prozessanzahl

**[Metricbeat] Abweichungen von der durchschnittlichen Prozessanzahl:** Eine andere Möglichkeit die Ausführung eines Kommandozeilenprogramms bzw. eher allgemein eines Programms zu erkennen erfolgt über die einfache Überwachung der Anzahl der Prozesse. So kann eine Abweichung von der durchschnittlichen Anzahl an Prozessen ein Indikator dafür sein, dass es zu nicht intendierten Aktionen im Cluster gekommen ist. Diese Erkennungsmethode ist vorwiegend im Vergleich mit der oben beschriebenen Methode nicht präzise. Das heißt, es gibt keine Auskunft über den gestarteten Prozess oder das Programm, aber es gibt zumindest ein Indiz, auf Basis dessen weitere Untersuchungen erfolgen können. In Hinblick auf den Fall, dass z.B. Falco ausfällt, besteht so weiterhin eine Möglichkeit der Erkennung auf die zurückgegriffen werden kann. Zu berücksichtigen ist hier auch das dynamische Umfeld eines Kubernetes Clusters. Es werden u.U. in hoher Zahl Prozesse erzeugt oder beendet. Es sollte deswegen die durchschnittliche Anzahl als Referenz herangezogen werden.

### Visualisierung/ Alarmierung

Metricbeat überträgt die Anzahl an Prozessen eines Dienstes in den zugehörigen Index der Elasticsearch-Datenbank. Berechnet man den durchschnittlichen Wert zu der Anzahl an Prozessen und vergleicht die je Zeiteinheit gemeldeten Werte damit, lässt sich ein Graph wie in Abbildung 4.5 erstellen. Ausschläge in einem solchen Graphen wie der markierte (11:23 Uhr, Abweichung +2,75) können auf eine Auffälligkeit, der nachgegangen werden sollte, hinweisen.



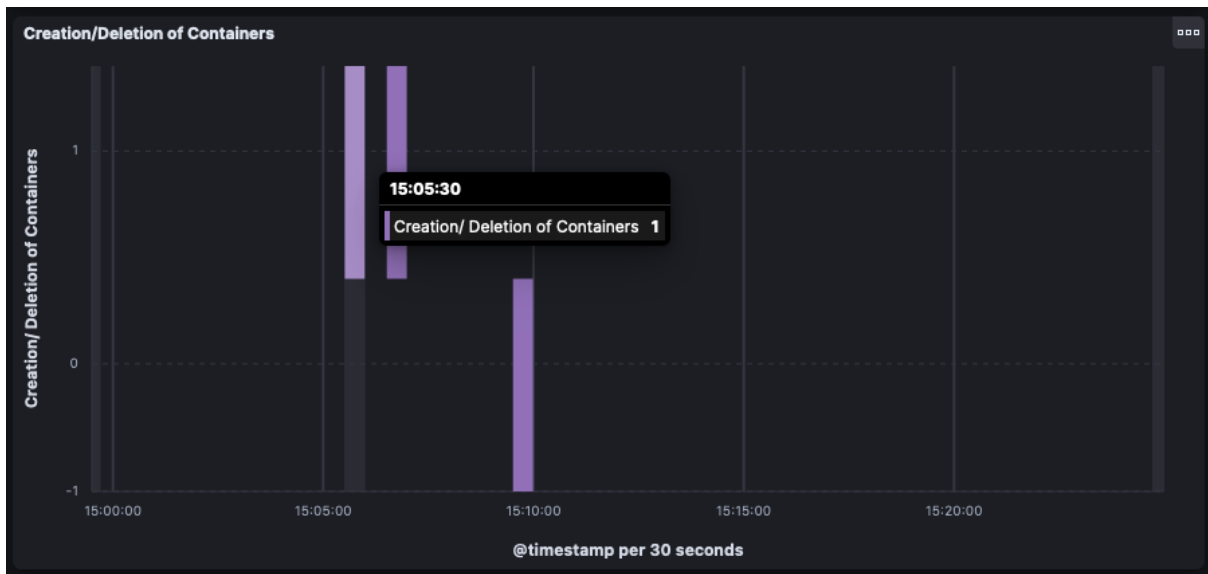


Abbildung 4.6: Kibana-Balkendiagramm: Erzeugung/ Löschung von Containern

## 4.2.2 Neuer Container (AU.3)

### Erkennungsansatz & Realisierung

Die Erkennung des Starts eines neuen Containers bzw. eines Pods (vgl. Bedrohung 'Neuer Container (AU.3)') lässt sich über zwei Wege realisieren, die einen unterschiedlichen Grad in der Qualität der Erkennung aufweisen und im Idealfall in Kombination eingesetzt werden sollten.

#### [Metricbeat] Veränderungen in der Anzahl an Containern:

Eine Möglichkeit die beschriebene Bedrohung zu erkennen bedient sich der Anzahl an Containern im Cluster und der Veränderung im Vergleich zum vorherigen Zustand. Über die Metricbeat-Komponente des Elastic-Stacks lassen sich Zustandsinformationen von der Container-Runtime in das Logging-System überführen. So auch die eindeutigen Identifizierungs-Zeichenketten aller existierenden Container. Die Anzahl dieser lässt sich zählen und im Falle einer Veränderung diese darstellen - Abbildung 4.6 zeigt eine mögliche Darstellung.

#### Visualisierung/ Alarmierung

In dem Balkendiagramm werden neu erzeugte Container als positiver ganzzahliger Wert dargestellt. Die markierte Stelle zeigt den Start eines neuen Containers um etwa 15:05 Uhr. Wird ein Container terminiert, wird dies als negativer Wert dargestellt.

Diese Darstellung ermöglicht im Mindestmaß eine Erkennung der Bedrohung. Es muss aber berücksichtigt werden, dass Kubernetes grundsätzlich ein dynamisches System ist: Ausgangspunkt für die automatische Erzeugung neuer Container ist z.B. die Möglichkeit des Autoscalings in Kubernetes über einen Horizontal-Pod-Autoscaler. Das wird in dieser Darstellungsweise nicht berücksichtigt und der manuelle Aufwand, dass neue Container untersucht werden müssen, die vom System erzeugt wurden, ist gegeben.

#### [Filebeat] Überwachung der Kubernetes-API:

Eine qualitativ hochwertigere Erfassung lässt sich durch Überwachung der Anfragen an die Kubernetes-API erreichen. Dazu werden über den Filebeat-Dienst die Log-Daten der Kubernetes-API in das Logging-System überführt und dort ausgewertet.

Die Kubernetes-API erstellt in der Standard-Konfiguration allerdings keine nutzbaren Log-Daten. Dies muss zunächst konfiguriert werden. Eine sogenannte 'Audit-Policy' legt dabei fest, was und mit welcher Detailtiefe als Log-Eintrag festgehalten werden soll. Listing 4.5 zeigt den für die Bedrohung relevanten Ausschnitt aus einer solchen Policy.

```
1 apiVersion: audit.k8s.io/v1
2 kind: Policy
3 # Don't generate audit events for all requests in RequestReceived stage.
4 omitStages:
5   - "RequestReceived"
6 rules:
7   # Log pod changes at RequestResponse level
8   - level: RequestResponse
9     resources:
10    - group: ""
11      # Resource "pods" doesn't match requests to any subresource of pods
12      # which is consistent with the RBAC policy.
13      resources: ["pods"]
14   # Log "pods/log", "pods/status" at Metadata level
15   - level: Metadata
16     resources:
17     - group: ""
18       resources: ["pods/log", "pods/status"]
19
20 ...
21
22 # A catch-all rule to log all other requests at the Metadata level.
23 - level: Metadata
24   # Long-running requests like watches that fall under this rule will not
25   # generate an audit event in RequestReceived.
26   omitStages:
27     - "RequestReceived"
```

Listing 4.5: Kubernetes-API Audit-Policy Ausschnitt (nach [The21a])

In Kubernetes sind Horizontal-Pod-Autoscaler in der Lage die Anzahl der Instanzen eines Deployments, eines Replicasets oder eines Statefulsets auf Basis von bestimmten Metriken (z. B. CPU-Nutzung der aktuell ausgeführten Instanzen) zu verändern. API-seitig erfolgt dies durch einen PATCH oder UPDATE Request. Die Erstellung eines Pods erfolgt durch einen CREATE Request.

Diese Anfragen lassen sich in den Logs kombiniert abfragen: Wurde ein Pod erstellt, aber keine zugehörigen Anfragen zur Veränderung an eine höher abstrahierte Komponente gesendet, kann davon ausgegangen werden, dass die Erstellung des Pods nicht durch einen Horizontal-Pod-Autoscaler erfolgte. Wurde der Pod nicht geplant erzeugt, liegt ein Indiz für eine Bedrohung vor.

## Visualisierung/ Alarmierung

Listing 4.6 zeigt, wie eine Abfrage in der Kibana-Query-Language (Abfragesprache in Kibana) aussehen kann, um eine Visualisierung zu erreichen.

```
1 kubernetes.labels.component : "kube-apiserver" and
2
3 (((objectRef.resource :
4   ("replicasets" or "deployments" or "statefulsets" ))
5   and objectRef.subresource : "scale"
6   and verb : ( "patch" or "update" ))
7
8 or (objectRef.resource : "pods"
9   and verb : "create"))
```

Listing 4.6: KQL-Abfrage: Autoscaling von Deployments

Die Abbildung 4.7 zeigt im unteren Diagramm das Ergebnis der Abfrage aus Listing

4.6. Es lässt sich ablesen, dass um etwa 15:05 Uhr ein Pod erzeugt wurde. Es wurde allerdings nicht wie bei dem zweiten Eintrag, eine PATCH- bzw. UPDATE-Anfrage an die Kubernetes-API erfasst (der zweite Eintrag ist durch einen Horizontal-Pod-Autoscaler erzeugt worden). Um besser feststellen zu können ob der manuelle Pod-Start gewollt war, zeigt das obere Diagramm den Namen des Pods an der erzeugt wurde. Bei dem um etwa 15:05 Uhr erzeugten Pod ist hier der Name 'evilpod'.

### 4.2.3 Schreibbarer Host-Pfad-Mount (PS.2)

#### Erkennungsansatz & Realisierung

Ein unintendiertes Einbinden von Dateien oder Verzeichnissen des Host-Systems kann ein großes Sicherheitsrisiko darstellen (vgl. Bedrohung 'Schreibbarer Host-Pfad-Mount (PS.2)'). Zur Erkennung von solchen Host-Pfad-Mounts wird zum einen auf eine Erkennung durch Falco und zum anderen auf die Analyse von Anfragen zur Erstellung von Workloads an die Kubernetes-API gesetzt.

#### [Falco] Einbindung kritischer Pfade in einen Container & Schreibzugriffe auf kritische Pfade:

Mit Falco lassen sich Einbindungen von kritischen Pfaden des Host-Systems in einen Container erfassen. Dazu gleicht Falco die 'Mount Destinations' respektive die Verzeichnisse, die in einen Container eingebunden werden, mit einer Liste von, als kritisch definierten, Verzeichnissen ab. In dieser Liste sind unter anderem Verzeichnisse wie '/etc' oder '/etc/kubernetes' standardmäßig spezifiziert. Die Liste lässt sich um individuelle Pfade erweitern. Listing 4.7 zeigt die für eine Erkennung relevante Regel.



Abbildung 4.7: Kibana-Diagramme: API basierte Überwachung von Pod Erzeugungen

```
1 - macro: sensitive_mount
2   condition: (container.mount.dest[/proc*] != "N/A" or
3               container.mount.dest[/var/run/docker.sock] != "N/A" or
4               container.mount.dest[/var/lib/kubelet] != "N/A" or
5               ...
6   ...
7
8 - rule: Launch Sensitive Mount Container
9   desc: >
10    Detect the initial process started by a container that has a mount from a sensitive
    host directory
11    (i.e. /proc). Exceptions are made for known trusted images.
12   condition: >
13    container_started and container
14    and sensitive_mount
15    and not falco_sensitive_mount_containers
16    and not user_sensitive_mount_containers
17   output: Container with sensitive mount started (user=%user.name
    user_loginuid=%user.loginuid command=%proc.cmdline %container.info
    image=%container.image.repository:%container.image.tag mounts=%container.mounts)
18   priority: INFO
19   tags: [container, cis, mitre_lateral_movement]
```

Listing 4.7: Falco-Regel: Einbindung kritischer Pfade

Die Gefahr, die von der Einbindung kritischer Pfade ausgeht, lässt sich wie folgt unterscheiden:

- Sollte eine Einbindung mit ausschließlicher Lese-Berechtigung erfolgen, besteht die Gefahr, dass Informationen unintendiert abfließen, die Gefahr von Manipulationen ist allerdings nicht gegeben. Dieser Fall kann von der oben angegebenen Regel erkannt werden.
- Erfolgt eine Einbindung mit Schreib-Berechtigungen besteht die Gefahr der Manipulation (z. B. Veränderung von Konfigurationen). Dieser Fall ist kritischer, als der reine Lese-Zugriff. Neben der grundsätzlichen Erkennung einer Einbindung sollte daher auch eine Erkennung von Schreib-Aktionen auf Dateien in kritischen Verzeichnissen erkannt werden. Eine solche Regel ist in Listing 4.8 exemplarisch für den Pfad '/etc' dargestellt.

```
1 - rule: Write below etc
2   desc: an attempt to write to any file below /etc (excl. common/ system applications)
3   condition: write_etc_common
4   output: "File below /etc opened for writing (user=%user.name user_loginuid=%user.loginuid
    command=%proc.cmdline parent=%proc.pname pcmdline=%proc.pcmdline file=%fd.name
    program=%proc.name gparent=%proc.aname[2] ggparent=%proc.aname[3] gggparent=%proc.aname[4]
    container_id=%container.id image=%container.image.repository)"
5   priority: ERROR
6   tags: [filesystem, mitre_persistence]
```

Listing 4.8: Falco-Regel: Schreib-Zugriff kritischer Pfad

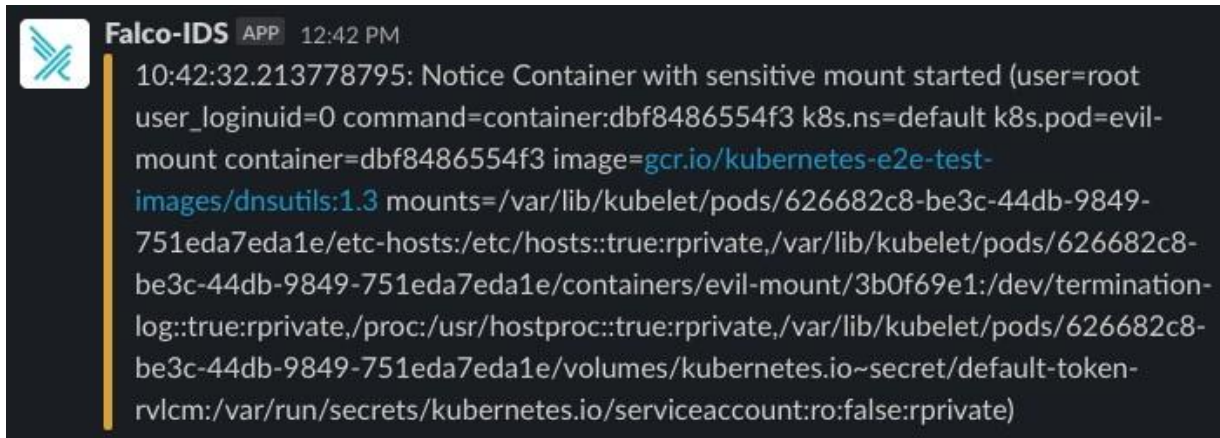


Abbildung 4.8: Slack-Nachricht: Falco Ereignis - kritische Einbindung

### Visualisierung/ Alarmierung

Wie bei der praktischen Umsetzung in 4.2.1 'bash/cmd innerhalb des Containers' erfolgt für die von Falco geprüften Regeln eine Alarmierung über eine Nachricht in dem Messaging-Dienst Slack (siehe Abbildung 4.8). Außerdem werden die Alarmierungen in einem Diagramm in Kibana über den Zeitverlauf hinweg visualisiert (vgl. Darstellung in Abbildung 4.4).

### [Filebeat] Überwachung der Kubernetes-API:

Neben der auf Host-Ebene erfolgenden Erkennung durch Falco lassen sich die Erzeugung von Workloads mit kritischen Einbindungen von Verzeichnissen auch durch die Überwachung der Kubernetes-API erkennen. Wird die Kubernetes-API so konfiguriert, dass bei Anfragen zur Erzeugung von Workloads der Request-Body (Teil der API Anfrage, der die eigentliche Nutzlast mit der Definition von z. B. einem zu erstellenden Deployment enthält) miterfasst wird, lässt sich dieser auf kritische Einbindungen hin untersuchen. In der Spezifizierung eines Workloads muss, um ein Verzeichnis des Hosts einzubinden, der Schlüssel '[...].spec.volumes.hostPath[...] ' vorhanden sein. Hiernach kann man die erfassten Anfragen durchsuchen. Der notwendige Teil der Kubernetes-API Audit-Log-Konfiguration ist exemplarisch für die Workloads der API-Gruppe 'apps' in Listing 4.9 angegeben.

```
1 - level: Request
2   resources:
3     - group: "apps"
4       resources: ["deployments", "replicasets", "statefulsets", "daemonsets"]
5       verbs: ["create", "patch", "update"]
```

Listing 4.9: Kubernetes-API Audit-Log-Konfiguration 'Apps'

### Visualisierung/ Alarmierung

In Kibana lässt sich ein Diagramm erzeugen, in welchem Treffer für die Überprüfung nach einem vorhandenen 'hostPath' Schlüssel dargestellt werden. Die Darstellung lässt sich wie in Abbildung 4.9 gezeigt durch die Spezifizierung des betroffenen Workload-Objektes und des eingebundenen kritischen Pfades anreichern. In dem gezeigten Diagramm wurde ein Deployment erzeugt, welches das Verzeichnis '/proc' des Hosts einbindet. Da das Deployment und die aus dem Deployment resultierende Replicaset-Instanz zwei, zwar verknüpfte, aber separate Objekte der Kubernetes-API sind, wird 'evil-mount' zweimal aufgeführt.



Abbildung 4.9: Kibana-Diagramm: Workload mit HostPath-Mount

#### 4.2.4 Kubernetes CronJob (PS.3)

##### Erkennungsansatz & Realisierung

Wie native CronJobs auf einem Linux-System, definieren Kubernetes-CronJobs Routinen, die in bestimmten Intervallen wiederholt werden. Dieser Umstand der regelmäßigen Ausführung kann von Angreifer:innen missbraucht werden (vgl. Bedrohung 'Kubernetes CronJob (PS.3)'). Die Erkennung dieser Bedrohung gliedert sich in zwei Bereiche:

- Die Erzeugung der CronJob-Ressource selbst. Dieser Teil ist der grundsätzlich kritischere Teil, da hier die initiale Grundlage gelegt wird für die eigentliche Ausführung in Jobs.
- Die automatische und vom im CronJob spezifizierten Intervall wiederholende Erzeugung von Jobs (Ausführungsinstanzen).

Konkret erfolgt die Erkennung durch Überwachung der Anfragen an die Kubernetes-API. Von Relevanz sind hier solche Anfragen, die einen neuen CronJob anlegen und solche, die die Erzeugung von Jobs darstellen (siehe auch Listing 4.10 für die notwendige Kubernetes-API Audit-Log-Konfiguration).

Created CronJobs			
Top values of objectRef.r	Timestamp per minute	Schedule	Namespace
evil-cronjob	14:43	*/* * * *	default

Abbildung 4.10: Kibana-Tabelle: Erzeugte CronJobs

```
1 - level: Request
2   resources:
3 -   group: "batch"
4     resources: ["cronjobs", "jobs"]
5     verbs: ["create", "patch", "update"]
```

Listing 4.10: Kubernetes-API Audit-Log-Konfiguration 'Batch'



## Visualisierung/ Alarmierung

Die Erzeugung eines neuen CronJobs ist der initiale kritische Schritt in diesem Bedrohungsszenario. In Kibana lässt sich eine Tabellen-Darstellung anlegen, die eine Übersicht über die erzeugten CronJobs gibt und mit zusätzlichen Informationen darstellt (siehe Abbildung 4.10).

Um eine Echtzeit-Alarmierung zu erreichen, können in Kibana Alarmierungsregeln definiert werden. Die Abbildung 4.11 zeigt eine mögliche Regel für die Erkennung von Erzeugungen von CronJob-Ressourcen. Die Regel wird standardmäßig alle 5 Minuten ausgeführt und prüft die letzten 6 Minuten darauf, ob es mindestens einen Eintrag passend zu den Filterkriterien gibt. Ist dies der Fall wird der Alarm ausgelöst und eine spezifizierte Aktion ausgeführt (z. B. Senden einer E-Mail, ergänzend zu Benachrichtigung durch Falco (das Senden dieser Nachrichten erforderten einen bezahlten Plan des Elastic Stacks)).

Um zusätzlich fortlaufend die Ausführung von CronJobs überwachen zu können, lässt sich ein Diagramm erzeugen, welches die Starts der Job-Instanzen über den Zeitverlauf darstellt (äquivalent mit der Darstellung von Pod Erzeugungen, vgl. Abbildung 4.7). Die Abbildung 4.12 zeigt ein solches Diagramm.

Die Bedrohung, die von der eigentlich im Job ausgeführten Anwendung respektive Routine ausgeht, wird hier nicht weiter betrachtet. Sie kann in vielen Fällen durch die in dem theoretischen Teil ausgeführten Maßnahmen sowie durch die im praktischen Teil gezeigten Maßnahmen erkannt werden.

### 4.2.5 Container-Logs löschen (UV.1)

#### Erkennungsansatz & Realisierung

Um zu erkennen, dass Log-Daten gelöscht werden respektive die Übertragung von Log Daten unterbrochen wurde (vgl. Bedrohung 'Container-Logs löschen (UV.1)'), werden

The screenshot displays the Kibana 'Index patterns' configuration interface. At the top, the 'Index patterns' section shows 'filebeat-\*' with a 'Reset to default index patterns' link. Below this, a text box explains that the pattern is used to filter Elasticsearch indices. The 'Custom query' section contains a query: 'objectRef.resource : \"cronjobs\" and verb : (\"create\" or \"patch\" or \"update\" ) and not objectRef.subresource : \"status\"'. Below the query, there are two 'Group by' rules. The first rule groups by 'objectRef.name' with a threshold of '1'. The second rule also groups by 'objectRef.name' with a threshold of '1'. The interface includes various interactive elements like dropdowns, buttons, and a 'KQL' button.

Abbildung 4.11: Kibana: Alarmierungsregel Erzeugung CronJob





Abbildung 4.12: Kibana-Diagramm: Ausführungen CronJobs

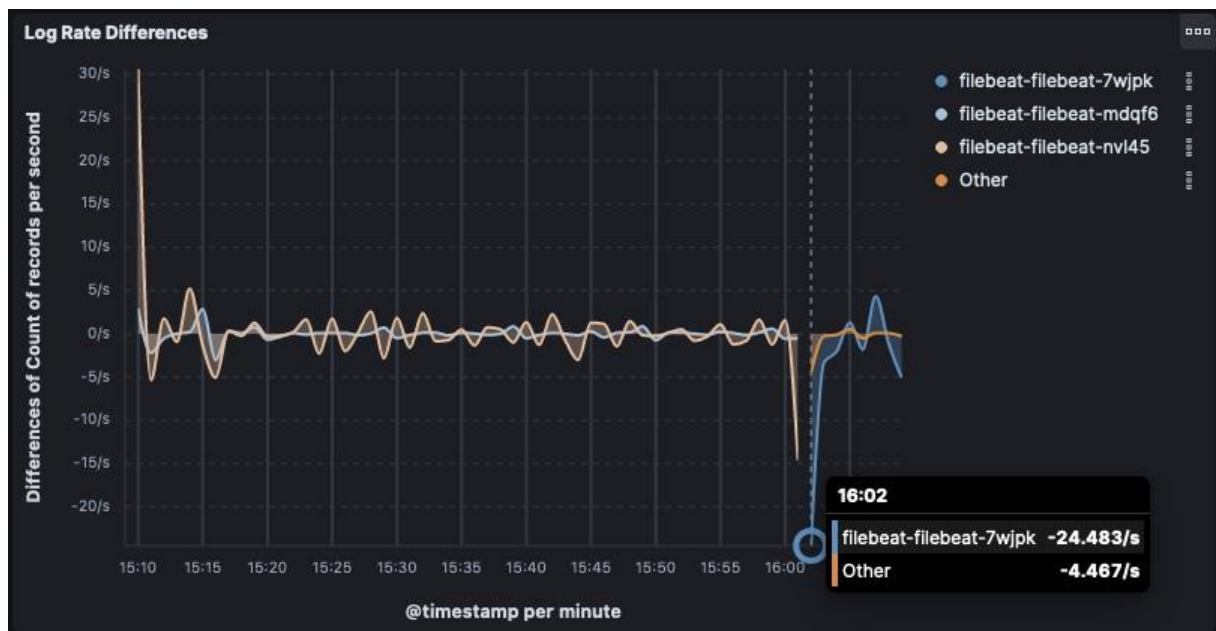


Abbildung 4.13: Kibana-Diagramm: Abweichungen der Log-Rate

zwei Ansätze realisiert. Die durch die zwei Ansätze erreichte Redundanz entspricht dem ‘defense-in-depth’-Prinzip und ermöglicht es zu erkennen, dass einer der beiden Wege kompromittiert wurde. Es wird hierzu die Log-Rate der Filebeat-Komponente überwacht sowie mithilfe von Falco versucht zu erkennen, ob Löschbefehle auf einem Host (oder aus einem Container heraus) ausgeführt werden, die Container Log Daten betreffen.

#### [Filebeat] Überwachung der Log-Rate:

Wie in ‘Container-Logs löschen (UV.1)’ beschrieben, ist es möglich in Kibana die Log-Rate, das heißt die Anzahl an eingehenden Log-Einträgen auf einen bestimmten Zeitraum bezogen darzustellen (vgl. Abbildung 3.2). Auf Basis dieser Information lässt sich auch eine Differenz dieser Rate von einem Mittelwert berechnen und darstellen. Große respektive auffällige Abweichungen deuten entsprechend auf besondere Ereignisse hin – wie z. B. die Manipulation des Log-Systems.

## Visualisierung/ Alarmierung

Abbildung 4.13 zeigt eine mögliche Darstellung des beschriebenen Ansatzes. In dem Diagramm wird die Abweichung der Log-Rate auf eine Sekunde normalisiert und je Filebeat-Instanz angezeigt. Da Kibana die zugrundeliegenden Daten in sogenannten Buckets zusammenfasst, entstehen am Rand des Graphen u.U. Ausschläge (siehe linker Rand im Diagramm). Diese können als nicht weiter auffällig betrachtet werden. Die grundsätzlich leichten Schwankungen im Diagramm sind systembedingt und ebenfalls nicht weiter auffällig. Zu beachten ist hingegen der Einbruch um etwa 16:00 Uhr. Hier wurde die Übertragung von Log Daten unterbrochen.

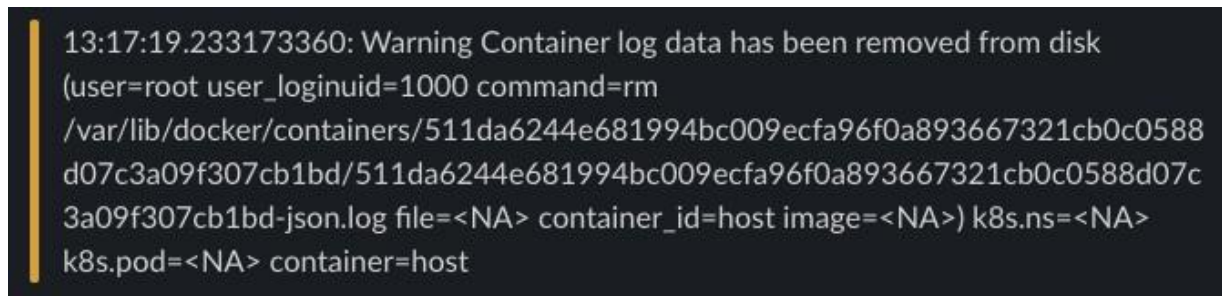


Abbildung 4.14: Slack-Nachricht: Falco Ereignis - Löschen von Container Logs

### [Falco] Meldung von Löschvorgängen:

Um konkrete Löschvorgänge von Container Log Daten zu erkennen, lässt sich für Falco die in Listing 4.11 angegebene Regel definieren. Damit ein Alarm ausgelöst wird, müssen folgende Bedingungen erfüllt sein:

- Es muss ein neuer Prozess gestartet worden sein,
- dieser Prozess gehört zu dem Aufruf eines bekannten Löschkommandos (z.B. 'shred' oder 'rm')
- und die betroffene Datei liegt in dem Verzeichnis '/var/lib/docker/containers'.

```
1 - rule: Delete Container Log Data
2   desc: Detect deletion of container log data
3   condition: spawned_process and clear_data_procs and fd.directory =
"/var/lib/docker/containers"
4   output: >
5     Container log data has been removed from disk (user=%user.name
user_loginuid=%user.loginuid command=%proc.cmdline file=%fd.name container_id=%container.id
image=%container.image.repository)
6   priority:
7     WARNING
8   tags: [process, filesystem]
```

Listing 4.11: Falco-Regel: Löschung von Container Log Daten

## Visualisierung/ Alarmierung

Erfolgt die Auslösung eines Alarms, wird eine Slack-Nachricht, mit entsprechenden Informationen zu dem ausgeführten Kommando und der betroffenen Datei, versendet (siehe Abbildung 4.14).

## 4.2.6 Netzwerk-Mapping bzw. Cluster-internes Networking (EK.3 bzw. LM.2)

### Erkennungsansatz & Realisierung

Die Erkennung von Bedrohungen für ein Kubernetes-Cluster durch die Analyse des Netzwerkverkehrs ist sehr effektiv und umfassend, allerdings auch komplex. In dieser praktischen Umsetzung der Maßnahmen zur Erkennung von netzwerkbasierenden Bedrohungen (vgl. 'Netzwerk-Mapping' (EK.3) und 'Cluster-internes Networking' (LM.2)), wird daher nur auf drei konkrete Fälle eingegangen.

#### [Packetbeat] Anfragen ungewollten Ursprungs an die Kubernetes-API:

Die Kubernetes-API als zentrale Steuerungskomponente des Clusters ist für ein:e Angreifer:in ein primäres Ziel. Scan-Programme wie 'kube-hunter' ([AP21]) können eingesetzt werden, um vorhandene Cluster zu entdecken und Informationen zu sammeln wie z. B. die Cluster-Version, aber auch um auf bekannte Schwachstellen hin zu testen.

Wird Packetbeat so konfiguriert, dass auf Node-Ebene der Netzwerkverkehr analysiert wird, lassen sich Anfragen ungewollten Ursprungs (z. B. von außerhalb des Clusters) an die Kubernetes-API erkennen. Es ist im Sinne des Zusammenspiels der anderen Sicherheitsbereiche (Prävention und Reaktion) zu empfehlen den Verkehr zu der Kubernetes-API zu beschränken. Bei der Verfolgung eines 'defense-in-depth' Ansatzes, ist zusätzlich eine Detektionsmaßnahme einzurichten.

Konkret lässt sich eine Erkennung umsetzen, in dem sämtlicher erfasster Netzwerkverkehr auf die Kubernetes-System-Ports:

- Kubernetes-API: 6443
- Kubelet-API: 10250
- Etcd: 2379

hin untersucht wird. Gleicht man die erfassten Pakete mit einer Liste an erlaubten IP-Adressen (z. B. IP-Adressen der Nodes und des Cluster-Overlay-Netzwerkes (häufig 10.244.0.0/16)) ab, lassen sich ungewollte Anfragen darstellen.

### Visualisierung/ Alarmierung

Abbildung 4.15 zeigt exemplarisch eine solche Darstellung. In dem Diagramm sind Anfragen einer nicht erlaubten IP-Adresse an die Kubernetes-API zu erkennen. Erzeugt wurden diese Anfragen durch das Ausführen eines Scans mit 'kube-hunter'.

#### [Filebeat] Verstöße gegen Firewall-Regeln:

Ein anderes Beispiel für eine Erkennung von auffälligem Netzwerkverkehr ist die Betrachtung von Verstößen gegen Firewall-Regeln. In dieser Umsetzung wurden konkrete Verstöße gegen Regeln der Uncomplicated Firewall (UFW) erfasst. Standardmäßig werden 'UFW-Blocks', also Fälle von gegen eine Regel verstoßen wurde und die Firewall blockiert hat, in den Linux-System-Logs eingetragen. Diese lassen sich mit Filebeat der Elasticsearch-Datenbank zuführen und in Kibana visualisieren. Es lassen sich hierdurch z. B. Port-Scans erkennen oder Versuche Anwendungen hinter für bestimmte Netzwerksegmente blockierten Ports zu erreichen.

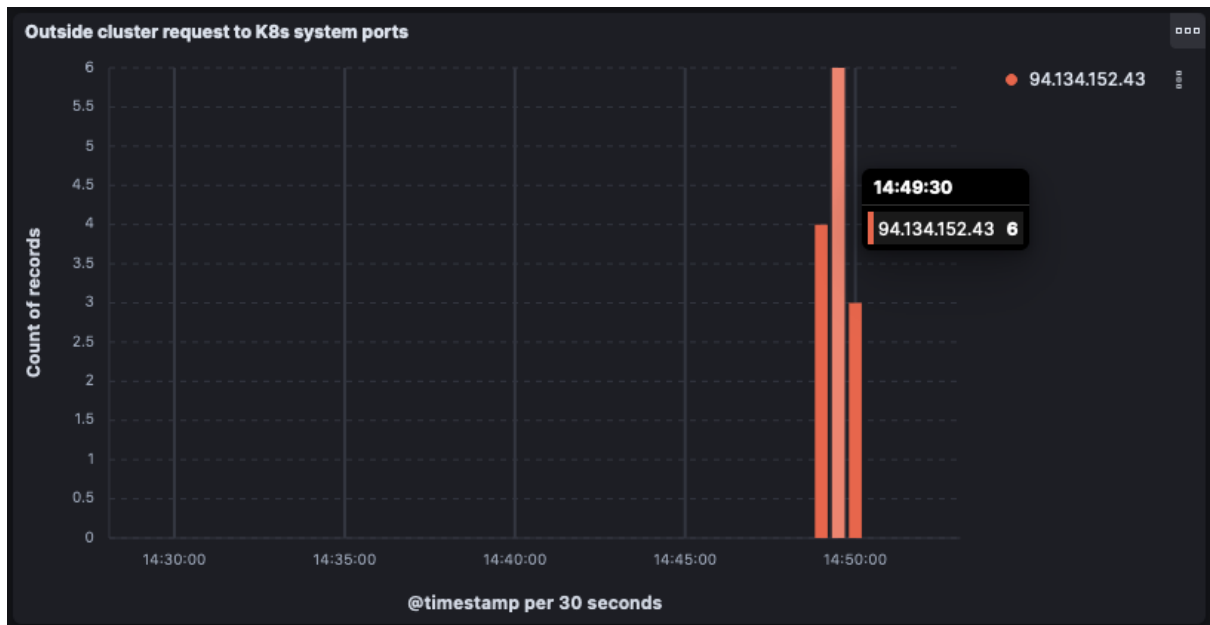


Abbildung 4.15: Kibana-Diagramm: Ungewollte Anfragen an Kubernetes-System-Ports

### Visualisierung/ Alarmierung

Die Abbildung 4.16 zeigt eine mögliche Darstellung der erkannten UFW-Blocks, aufgeteilt je Node des Clusters. An der markierten Stelle (14:37 Uhr) sind ungewöhnlich viele Verstöße erkannt worden. Dies stellt eine Auffälligkeit dar. Der Ausschlag wurde durch die Verwendung des Port-Scanners 'nmap' erreicht.

#### [Packetbeat] Ungewöhnlicher ausgehender Netzwerkverkehr:

Um z. B. den Abfluss an Informationswerten zu erkennen, ist es interessant den ausgehenden Netzwerkverkehr auf ungewöhnliche Ziel-IP-Adressen hin zu untersuchen. Packetbeat erfasst die Ziel-Adressen der erfassten Pakete. Dadurch ist es möglich eine Auflistung der häufigsten Ziel-IP-Adressen, je Zeitraum zu erstellen. Legitime Adressen bzw. Subnetze lassen sich hier ausschließen.

### Visualisierung/ Alarmierung

Eine mögliche Darstellung ist in Abbildung 4.17 gezeigt. In dem Diagramm wird die Menge des Netzwerkverkehrs zu einer ungewöhnlichen Ziel-IP-Adresse visualisiert. Exemplarisch wurden hier zwei Adressen hervorgehoben.

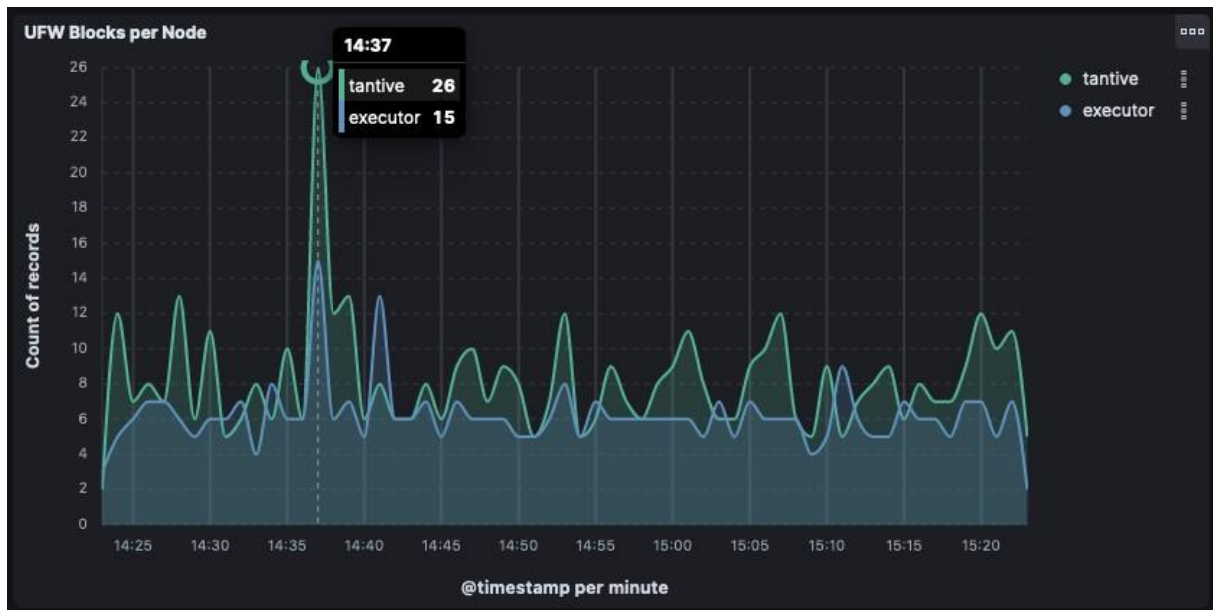


Abbildung 4.16: Kibana-Diagramm: UFW-Blocks je Node

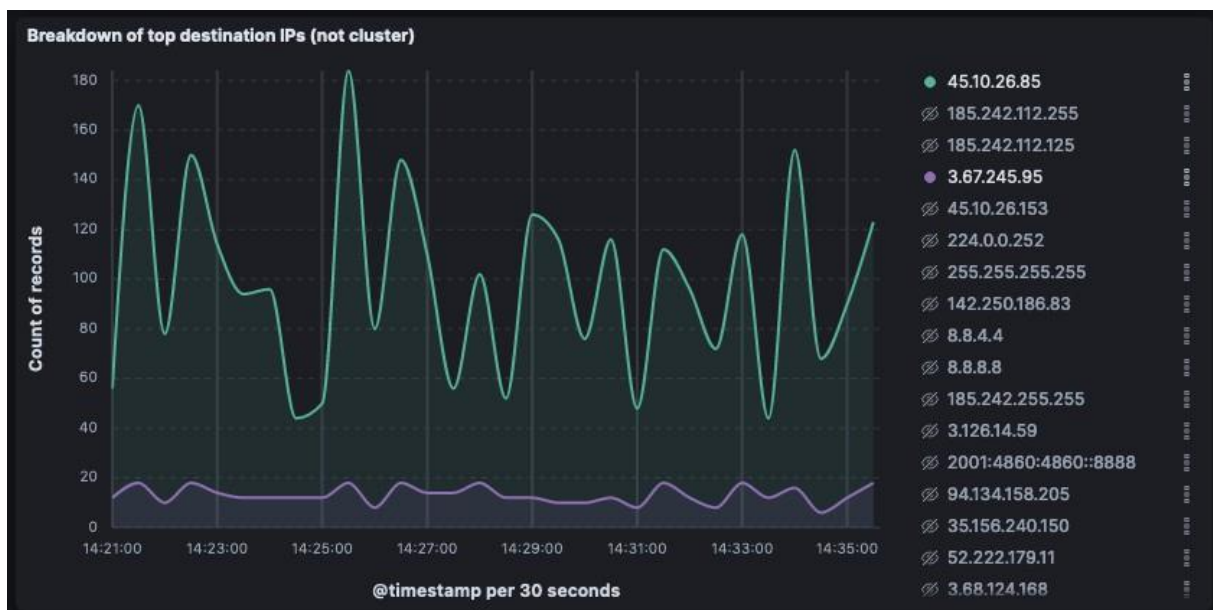


Abbildung 4.17: Kibana-Diagramm: Top Ziel-IP-Adressen

## 5 Evaluierung

Im Folgenden wird auf Basis der praktischen Umsetzungen geprüft, ob die in Kapitel 3 theoretisch entwickelten Ansätze zur Erkennung der verschiedenen Angriffstypen (vgl. Kubernetes Bedrohungsmatrix, Abbildung 3.1) wirksam sind. In Kapitel 4 wurden repräsentativ einige theoretisch vorgestellte Maßnahmen praktisch umgesetzt.

Stellvertretend für den Angriffstyp ‘Ausführung’ wurden Maßnahmen zur Erkennung der Ausführung von Kommandozeilenprogrammen innerhalb von Containern des Clusters realisiert (vgl. praktische Umsetzung, ‘bash/cmd innerhalb des Containers (AU.2)’). Die vorgestellten Erkennungsmechanismen lassen sich auf eine grundlegende Erfassung der Ausführung von Programmen generalisieren. Durch die Erfassung auf Basis von Charakteristika (vgl. Abschnitt ‘[Falco] Live-Erkennung über Syscalls und Falco-Engine’), die typisch für eine Programmklasse sind, besteht die Möglichkeit für aktuell nicht betrachtete Programmklassen ebenfalls eine solche Erkennungsmöglichkeit zu schaffen. So ist es mit dem Mechanismus grundsätzlich auch möglich die Ausführung von z. B. Webshells oder unerwartete Verbindungen zu erkennen. Zusätzlich wurde gezeigt, wie auf Basis von Daten der Metaebene (vgl. ‘[Metricbeat] Abweichungen von der durchschnittlichen Prozessanzahl’) auffällige Muster genutzt werden können, um ungewollte Ausführungen von Programmen zu erkennen. Dieses Vorgehen lässt sich ebenfalls erweitern. Z. B. indem die betrachtete Metrik geändert wird (z. B. Anzahl an Sockets oder mit dem Node verbundener Geräte anstelle von Prozessen).

Maßnahmen, um zu erkennen, ob ein neuer Container gestartet wurde, sind im Abschnitt ‘Neuer Container (AU.3)’ des Kapitels 4 exemplarisch umgesetzt worden. Hierbei wurde gezeigt, wie sich andere (Management-) Vorgänge im Cluster (z. B. die Ausführung von Jobs zur Erneuerung von TLS Zertifikaten, für die ein Pod gestartet wird oder der Ablauf von Sicherungsaufgaben) mit einem grundlegenden Ereignis korrelieren lassen, um damit die Qualität der Erkennung zu erhöhen respektive eine Erkennung zu ermöglichen (vgl. Korrelieren des Autoscalings und der Erzeugung eines Pods, Abschnitt ‘[Filebeat] Überwachung der Kubernetes-API’). Dieses Vorgehen kann exemplarisch für eine Vielzahl von weiteren Einsatzszenarien gesehen werden.

Die umgesetzten Maßnahmen der konkreten Bedrohungsszenarien ‘bash/cmd innerhalb des Containers (AU.2)’ und ‘Neuer Container (AU.3)’ lassen sich also gut generalisieren

und auf weitere Fälle anderer Angriffstypen wie z. B. ‘Sidecar Injection (AU.5)’ oder aber mit Bezug auf die eingeführte Kubernetes-API Überwachung auch auf z. B. ‘Cluster-Admin-Binding (PE.2)’ übertragen.

Vergleichbar stellvertretend wurde durch die Umsetzung von Maßnahmen zur Erkennung der Bedrohung ‘Schreibbarer Host-Pfad-Mount (PS.2)’ gezeigt, dass Angriffe der Typen ‘Persistence’, ‘Privilegien Eskalation’ sowie ‘Lateral Movement’ mit den gezeigten Ansätzen grundsätzlich detektierbar sind. Der erbrachte praktische Nachweis beschränkt sich hier auf solche Maßnahmen, die eine Erkennung auf Basis des Dateisystems der Nodes und Überwachung der Kubernetes-API erreichen. Der grundsätzliche Ansatz der Überwachung des Dateisystems und der Kubernetes-API lässt sich allerdings auf Anwendungsfälle wie z. B. die Erkennung von Vernichtung von Anwendungsdateien respektive Nutzdaten erweitern. Ergänzt wird dieser Nachweis durch die Umsetzung der Maßnahmen im Zusammenhang mit ‘Kubernetes-CronJobs (PS.3)’.

Der Angriffstyp ‘Umgehung der Verteidigung’ wird durch die Umsetzung der Erkennungsmaßnahmen der Bedrohung ‘Container-Logs löschen (UV.1)’ repräsentiert. Hier wurde gezeigt, dass die im theoretischen Teil aufgeführten Maßnahmen (z. B. Überwachung der Log-Rate) effektiv auf auffälliges Verhalten hinweisen können. Es wurde nachgewiesen, dass eine Überwachung des Logging-Systems selbst möglich ist und so Versuche, die Detektion durch dieses zu umgehen, erkannt werden können. In Verbindung mit Erkennungsmechanismen, die auf mit Falco umgesetzten Verfahren respektive auf der Kubernetes-API Überwachung aufbauen, lassen sich auch weitere im theoretischen Teil betrachtete



Bedrohungen detektieren (z. B. durch die API- Überwachung das Löschen von Kubernetes-Ereignissen).

Bei allen betrachteten Angriffstypen gibt es Bedrohungen, die einen starken Netzwerkbezug haben. Repräsentativ für diese wurde in Abschnitt 'Netzwerk-Mapping bzw. Cluster-internes Networking (EK.3 bzw. LM.2)' praktisch betrachtet, wie solche netzwerkbasierten Bedrohungen erkennbar gemacht werden können. Dabei wurde gezeigt, dass eine Analyse des gesamten Netzwerkverkehrs des Clusters möglich ist und auf Basis dessen verschiedenste Auffälligkeiten erkannt werden können. Die konkret betrachteten Fälle (z. B. Zugriffsversuche auf die Kubernetes-API von außen oder die Überwachung allgemeinen Netzwerkverkehrsverhaltens) verdeutlichen, wie umfassend der Netzwerkverkehr analysierbar ist. Die dargestellte Vorgehensweise macht jeglichen Netzverkehr innerhalb des Clusters respektive zwischen Cluster und Außenumgebung über das Monitoringsystem zugänglich. Daher wird es also möglich, existierenden Lösungen zur Netzverkehrsüberwachung angemessen auf Kubernetes-Cluster und die darin orchestrierten Systeme anzuwenden.

Grundsätzlich sind die im theoretischen Teil aufgezeigten Ansätze realistisch umsetzbar und effektiv wirksam. Es gilt allerdings zu beachten, dass durch die reduzierte repräsentative Auswahl von Bedrohungen, nur eine generalisierte Evaluierung für die effektive Wirksamkeit aller Maßnahmen möglich ist. So gibt es einzelne Bedrohungsszenarien, die von den Repräsentanten weniger gut mit eingefasst werden (z. B. 'Kompromittierte Images in der Registry (IZ.1)'). Für diese ist der Wert der hier getroffenen Aussage der Wirksamkeit entsprechend geringer und weitergehende Überprüfungen sind notwendig, aber nicht in Rahmen dieser Arbeit leistbar.

## 6 Zusammenfassung und Fazit

Aktuell werden Anwendungen zunehmend in der Entwicklung genauso wie im Produktionsbetrieb containervirtualisiert ausgeführt. Um die häufig große Anzahl an Containern, die Dynamik von modernen, virtualisierten Infrastrukturen und weitergehende Administrationsaufgaben wie Skalierung gut verwalten zu können, werden Orchestrierungssysteme wie Kubernetes eingesetzt. Mit dem Einsatz dieser Systeme muss auch deren Sicherheit entsprechend thematisiert und berücksichtigt werden. Klassische Sicherheitsmechanismen der Bereiche Prävention, Detektion und Reaktion müssen angepasst und angewendet werden.

Diese Arbeit betrachtet die Möglichkeiten und konkreten Maßnahmen zur Detektion von spezifischen Bedrohungen einer Kubernetes-orchestrierten Container-Infrastruktur. Ziel der Arbeit ist es, dass Entwerfen einer Logginginfrastruktur und eines Loggingkonzepts für die Detektion von Angriffen, durch eine strukturierte Modellierung von Angriffsszenarien und durch Aufzeigen von generellen aber auch sehr konkreten Maßnahmen, allgemein zu erleichtern.

Um dieses Ziel zu erreichen wurden auf Basis eines Bedrohungsmodells für Kubernetes spezifische Angriffe identifiziert und analysiert (vgl. Kapitel 3). Es wurde betrachtet, welche Gefahr von den ausgewählten Bedrohungen ausgeht und wie diese durch Loggingmaßnahmen zu erkennen sind. Dazu wurde bestimmt, welche Ansätze möglich sind und diese dann durch konkrete Maßnahmen angereichert. Es ergibt sich so ein umfassender Überblick über zu berücksichtigende Angriffstypen und eine Hilfestellung bei dem Aufbau eines Loggingsystems zur Erkennung von Bedrohungen.

Sollten allerdings die in dieser Arbeit entwickelten Ansätze und Erkennungsmechanismen in der Entwicklung eines Loggingsystems angewendet werden, sind entsprechend alle Bedrohungen zu berücksichtigen und nicht nur die, welche in dieser Arbeit praktisch umgesetzt wurden. Ebenso wird an dieser Stelle darauf hingewiesen, dass die betrachteten Bedrohungsszenarien zwar auf den Anspruch der Vollständigkeit ausgerichtet sind, diesem aber aufgrund der Vielzahl an möglichen Bedrohungsvarianten nicht umfänglich gerecht werden. Vielmehr ist eine fortlaufende Erweiterung und Berücksichtigung neuer relevanter Bedrohungen vorzunehmen.

Auf Basis des aktuellen Stands der verwendeten Bedrohungsmatrix wurden Ansätze entwickelt, die zum Zeitpunkt des Verfassens dieser Arbeit den wesentlichen Teil relevanter Bedrohungen berücksichtigen. Es wird verdeutlicht mit welchen Strukturen und Ansätzen vorgegangen werden kann und die Ergebnisse dieser Arbeit können dem Ziel entsprechend helfen sowie Orientierung geben, wenn ein:e Leser:in vor der Herausforderung des Aufbaus einer Logginginfrastruktur für ein Kubernetes-System steht.



## 7 Literaturverzeichnis

- [AP21] Aqua Security Software Ltd.; Project Contributors: GitHub- aquasecurity/kube-hunter: Hunt for security weaknesses in Kubernetes clusters. <https://github.com/aquasecurity/kube-hunter>. Version: 2021, Abruf: 2021-11-01
- [CBB+20] Cloud Security Alliance ; Brook, Jon-Michael ; Bhat, Suhas ; Begum, Calguner ; Eliyahu, Tal ; Getzin, Alex ; Hargrave, Vic ; Osmine, Ebudo ; Roza, Michael ; Yeoh, John ; Yousif, Nabeel: Top Threats to Cloud Computing. In: Top Threats to Cloud Computing: Egregious Eleven (2020), 30. <https://cloudsecurityalliance.org>
- [CG20] Coldwater, Ian; Geesaman, Brad: Advanced Persistence Threats: The Future of Kubernetes Attacks - Ian Coldwater & Brad Geesaman - YouTube. <https://www.youtube.com/watch?v=auUgVullAWM>. Version: 2020
- [com21] The kernel d. community: System Calls – The Linux Kernel documentation. <https://linux-kernel-labs.github.io/refs/heads/master/lectures/syscalls.html>. Version: 2021, Abruf: 2021-10-21
- [CSP13] Chuvakin, Anton ; Schmidt, Kevin ; Phillips, Chris: What is a Log? In: Logging and Log Management (2013), jan, S. 29–49. <http://dx.doi.org/10.1016/B978-1-59-749635-3.00002-6>. DOI 10.1016/B978-1-59-749635-3.00002-6. ISBN 978-1-59749-635-3
- [Doc21a] Docker Inc.: Docker Documentation | Glossary. <https://docs.docker.com/glossary>. Version: 2021, Abruf: 2021-08-17
- [Doc21b] Docker Inc.: What is a Container? | App Containerization | Docker <https://www.docker.com/resources/what-container>. Version: 2021, Abruf: 2021-08-17
- [Ela21a] Elasticsearch B.V: Configure the internal queue/ Filebeat Reference [7.15]/ Elastic. <https://www.elastic.co/guide/en/beats/filebeat/current/configuring-internal-queue.html>. Version: 2021, Abruf: 2021-10-14
- [Ela21b] Elasticsearch B.V: Data in: documents and indices/ Elasticsearch Guide [7.15]/ Elastic. <https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html> Version: 2021, Abruf: 2021-10-13
- [Ela21c] Elasticsearch B.V: Elastic Stack: Elasticsearch, Kibana, Beats und Logstash | Elastic. <https://www.elastic.co/de/elastic-stack/>. Version: 2021, Abruf: 2021-10-13
- [Ela21d] Elasticsearch B.V: Filebeat overview | Filebeat Reference [7.15] | Elastic. <https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html> {#}filebeat-overview. Version: 2021, Abruf: 2021-10-14
- [Ela21e] Elasticsearch B.V: Metricbeat overview/ Metricbeat Reference [7.15]/ Elastic. <https://www.elastic.co/guide/en/kibana/current/introduction.html>. Version: 2021, Abruf: 2021-10-13
- [Ela21f] Elasticsearch B.V: Metricbeat overview | Metricbeat Reference [7.15] | Elastic. <https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-overview.html> Version: 2021, Abruf: 2021-10-14
- [Ela21g] Elasticsearch B.V: Packetbeat overview | Packetbeat Reference [7.15] | Elastic. <https://www.elastic.co/guide/en/beats/packetbeat/current/packetbeat-overview.html>. Version: 2021, Abruf: 2021-10-14
- [Goo18] Gooding, Dan: Backdoored images downloaded 5 million times finally removed from Docker Hub | Ars Technica.. <https://arstechnica.com/information->

- technology/2018/06/backdoored-images-downloaded-5-million-times-finally-removed-from-docker-hub/. Version: 2018, Abruf: 2021-09-05
- [IS18] Indrasiri, Kasun ; Siriwardena, Prabath: Microservices for the Enterprise: Designing, Developing, and Deploying. San Jose, CA, USA : Apress, Berkeley, CA, 2018. – 422 S. – ISBN 1484238575
- [JRA14] Jouini, Mouna ; Rabai, Latifa Ben A. ; Aissaa, Anis B.: Classification of Security Threats in Information Systems | Elsevier Enhanced Reader. <http://dx.doi.org/10.1016/j.procs.2014.05.452>. Version: 2014, Abruf: 2021-08-30
- [KSN06] Kent, Karen; Souppaya, Murugiah; NIST: Special Publication 800-92 Guide to Computer Security Log Management Recommendations of the National Institute of Standards and Technology. (2006). <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf>
- [Kub21] Kube Notary Community: GitHub - codenotary/kube-notary: A Kubernetes watchdog for verifying image trust with CodeNotary. <https://github.com/codenotary/kube-notary>. Version: 2021, Abruf: 2021-09-08
- [Lub] Lubeck, Luis: Was ist MITRE ATT&CK und wozu dient es? | WeLiveSecurity. <https://www.welivesecurity.com/deutsch/2019/09/03/mitre-att-ck-framework/>, Abruf: 2021-09-05
- [Luk18] Lukša, Marko: Einführung in Kubernetes. Version: jul 2018. <http://dx.doi.org/10.3139/9783446456020.001>. In: Kubernetes in Action. München : Carl Hanser Verlag GmbH & Co. KG, jul 2018. – DOI 10.3139/9783446456020.001, 1–60
- [Mah20] Mahn, Jan: Kubernetes-Entwickler warnen vor internen Man-in-the-Middle-Angriffen | heise online <https://www.heise.de/news/Kubernetes-Entwickler-warnen-vor-internen-Man-in-the-Middle-Angriffen-4984747.html>. Version: 2020, Abruf: 2021-09-05
- [MW20] Microsoft ; Weizman, Yossi: Threat matrix for Kubernetes. <https://www.microsoft.com/security/blog/2020/04/02/attack-matrix-kubernetes/>. Version: 2020, Abruf: 2021-08-07
- [NC21] National Security Agency ; Cybersecurity and Infrastructure Security Agency: Kubernetes Hardening Guidance. (2021). [https://media.defense.gov/2021/Aug/03/2002820425/-1/-1/1/CTR\\_KUBERNETESHARDENINGGUIDANCE.PDF](https://media.defense.gov/2021/Aug/03/2002820425/-1/-1/1/CTR_KUBERNETESHARDENINGGUIDANCE.PDF)
- [OR21] Ovens, Steve ; Red Hat Inc.: The 7 most used Linux namespaces | Enable Sysadmin. <https://www.redhat.com/sysadmin/7-linux-namespaces>. Version: 2021, Abruf: 2021-08-17
- [Pas21] Passler: What is Syslog? <https://www.paessler.com/it-explained/syslog>. Version: 2021, Abruf: 2021-09-03
- [PF21] Padilla, Elmar ; Frauenhofer FKIE: Logs | Practical Digital Forensics. In: Vorlesung Digitale Forensik Hochschule Bonn-Rhein-Sieg WS 20/21 (2021)
- [PI82] Plummer, David ; IETF: rfc826. <https://datatracker.ietf.org/doc/html/rfc826>. Version: 1982, Abruf: 2021-09-27
- [Pro21] Project Contributors: GitHub - GoogleContainerTools/distroless: Language focused docker images, minus the operating system. <https://github.com/GoogleContainerTools/distroless>. Version: 2021, Abruf: 2021-09-09
- [Red21] Red Hat Inc.: State of Kubernetes Security Report. (2021). <https://www.redhat.com/rhdc/managed-files/cl-state-kubernetes-security-report-ebook-f29117-202106-en.pdf>

- [Ric17] Rice, Liz: A beginner's guide to Linux syscalls | Opensource.com. <https://opensource.com/article/17/5/beginners-guide-syscalls>. Version: 2017, Abruf: 2021-10-21
- [RT20] Remillano, Augusto; Trend Micro Incorporated: Malicious Docker Hub Container Images Used for Cryptocurrency Mining - Security News. <https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/malicious-docker-hub-container-images-cryptocurrency-mining>. Version: 2020, Abruf: 2021-09-06
- [SA19] Sagi, Daniel; Aqua Security Software Ltd.: DNS Spoofing on Kubernetes Clusters. <https://blog.aquasec.com/dns-spoofing-kubernetes-clusters>. Version: 2019, Abruf: 2021-09-27
- [SMMR16] Safina, Larisa ; Mazzara, Manuel ; Montesi, Fabrizio ; Rivera, Victor: Data-driven workflows for microservices: Genericity in jolie. In: Proceedings - International Conference on Advanced Information Networking and Applications, AINA 2016-May (2016), S. 430-437. <http://dx.doi.org/10.1109/AINA.2016.95>. – DOI 10.1109/AINA.2016.95. – ISBN 9781509018574
- [TD20] Tafani-Dereeper, Christophe: Privilege Escalation in AWS Elastic Kubernetes Service (EKS) by compromising the instance role of worker nodes - Christophe Tafani-Dereeper. <https://blog.christophetd.fr/privilege-escalation-in-aws-elastic-kubernetes-service-eks-by-compromising-the-instance-role-of-worker-nodes/>. Version: 2020, Abruf: 2021-09-05
- [The10] The CEE Editorial Board: Common Event Expression Architecture Overview Version 0.5. (2010). <https://cee.mitre.org/docs/CEE{ }Architecture{ }Overview-v0.5.pdf>
- [The21a] The Kubernetes Authors: Auditing | Kubernetes. <https://kubernetes.io/docs/tasks/debug-application-cluster/audit/> Version: 2021, Abruf: 2021-09-22
- [The21b] The Kubernetes Authors: Debugging with an ephemeral debug container | Monitoring, Logging, and Debugging | Kubernetes. <https://kubernetes.io/docs/tasks/debug-application-cluster/print/#ephemeral-container>. Version: 2021, Abruf: 2021-09-08
- [The21c] The Kubernetes Authors: Deploy and Access the Kubernetes Dashboard | Kubernetes. <https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/> Version: 2021, Abruf: 2021-09-08
- [The21d] The Kubernetes Authors: kube-apiserver | Kubernetes. <https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/> Version: 2021, Abruf: 2021-08-30
- [The21e] The Kubernetes Authors: Kubernetes Components | Kubernetes. <https://kubernetes.io/docs/concepts/overview/components/> Version: 2021, Abruf: 2021-08-30
- [The21f] The Kubernetes Authors: Pods | Kubernetes. <https://kubernetes.io/docs/concepts/workloads/pods/#using-pods> Version: 2021, Abruf: 2021-09-09
- [The21g] The Kubernetes Authors: Using RBAC Authorization | Kubernetes. <https://kubernetes.io/docs/reference/access-authn-authz/rbac/#privilege-escalation-prevention-and-bootstrapping> Version: 2021, Abruf: 2021-09-13
- [The21h] The Kubernetes Authors: Using RBAC Authorization | Kubernetes. <https://kubernetes.io/docs/reference/access-authn-authz/rbac/> Version: 2021, Abruf: 2021-09-21

- [The21i] The Kubernetes Authors: What is Kubernetes? | Kubernetes.  
<https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/> Version: 2021, Abruf: 2021-08-17
- [The21j] The MITRE Corporation: MITRE ATT&CK® <https://attack.mitre.org/> Version: 2021, Abruf: 2021-09-05
- [The21k] The Project Contributors: GitHub - opsgenie/kubernetes-event-exporter: Export Kubernetes events to multiple destinations with routing and filtering.  
<https://github.com/opsgenie/kubernetes-event-exporter> Version: 2021, Abruf: 2021-09-17
- [The21l] The Kubernetes Authors: The Kubernetes network model;  
<https://kubernetes.io/docs/concepts/cluster-administration/networking/#kubernetes-model> Version 2021, Abruf: 2021-01-01
- [TT21a] The Falco Authors ; The Linux Foundation: Rules | Falco. <https://falco.org/docs/rules/> Version: 2021, Abruf: 2021-10-21
- [TT21b] The Falco Authors ; The Linux Foundation: Supported Syscall Events | Falco.  
<https://falco.org/docs/rules/supported-events> Version: 2021, Abruf: 2021-10-21
- [TT21c] The Falco Authors; The Linux Foundation: The Falco Project | Falco. <https://falco.org/docs/> Version: 2021, Abruf: 2021-10-21
- [Vaa21] Vaas, Lisa: Microsoft: Big Cryptomining Attacks Hit Kubeflow | Threatpost.  
<https://threatpost.com/microsoft-cryptomining-kubeflow/166777/> Version: 2021, Abruf: 2021-09-05
- [VSTK19] Vayghan, Leila A. ; Saied, Mohamed A. ; Toeroe, Maria ; Khendek, Ferhat: Kubernetes as an Availability Manager for Microservice Applications. (2019). <http://arxiv.org/abs/1901.04946>
- [WM21] Weizman, Yossi; Microsoft Corporation: Secure containerized environments with updated threat matrix for Kubernetes | Microsoft Security Blog  
<https://www.microsoft.com/security/blog/2021/03/23/secure-containerized-environments-with-updated-threat-matrix-for-kubernetes/> Version: 2021, Abruf: 2021-09-05

## 8 Abbildungsverzeichnis

- 2.1 Entwicklung der Virtualisierung/ Isolierung 10
- 2.2 Grobe Struktur von Kubernetes 13
- 2.3 Definition Log 17
- 2.4 Dezentrales Logging 17
- 2.5 Zentralisiertes Logging 19
- 2.6 Schematische Struktur des Elastic-Stacks 21
  
- 3.1 Bedrohungsmatrix für Kubernetes (nach [WM21]) 25
- 3.2 Kibana: Log-Rate pro Minute 40
  
- 4.1 Grundlegende Infrastruktur 51
- 4.2 Übersicht der Logging Infrastruktur 55
- 4.3 Slack-Nachricht: Falco Ereignis - Start Kommandozeilenprogramm 56
- 4.4 Kibana-Balkendiagramm: Falco Ereignisse 56
- 4.5 Kibana-Balkendiagramm: Abweichung von durchschnittlicher Prozessanzahl 57
- 4.6 Kibana-Balkendiagramm: Erzeugung/ Löschung von Containern 58
- 4.7 Kibana-Diagramme: API basierte Überwachung von Pod Erzeugungen 61
- 4.8 Slack-Nachricht: Falco Ereignis - kritische Einbindung 63
- 4.9 Kibana-Diagramm: Workload mit HostPath-Mount 64
- 4.10 Kibana-Tabelle: Erzeugte CronJobs 64
- 4.11 Kibana: Alarmierungsregel Erzeugung CronJob 65
- 4.12 Kibana-Diagramm: Ausführungen CronJobs 66
- 4.13 Kibana-Diagramm: Abweichungen der Log-Rate 66
- 4.14 Slack-Nachricht: Falco Ereignis - Löschen von Container Logs 67
- 4.15 Kibana-Diagramm: Ungewollte Anfragen an Kubernetes-System-Ports 69
- 4.16 Kibana-Diagramm: UFW-Blocks je Node 70
- 4.17 Kibana-Diagramm: Top Ziel-IP-Adressen 70