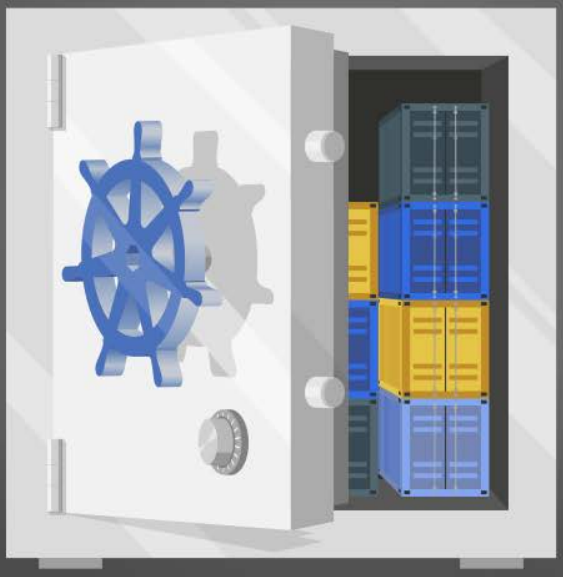
Tutorial: Kubernetes absichern – Episode 1

Ein sicherer Kubernetes-Betrieb erfordert einigen Aufwand und den Einsatz zusätzlicher Werkzeuge. Dann kann eine K8s-Umgebung aber deutlich mehr Sicherheit bieten als eine traditionelle IT.

Von Christoph Puppe



Wer Container sagt, meint heute fast immer Kubernetes (K8s). Es ist der Motor, der die Container-Revolution antreibt. Fast niemand nutzt es pur, die meisten verwenden eine Distribution, die den Einsatz des in der Rohform doch recht sperrigen Werkzeugs vereinfacht. Für On-Premises-Installationen teilen sich OpenShift, Rancher, Kubermatic, Tanzu und Docker Kubernetes Service den Markt. In der Cloud sind es neben den üblichen Verdächtigen AWS, Azure und GCP auch lokale Anbieter wie Plusserver, Ionos, Hetzner und weitere, die Managed Kubernetes anbieten.

Die allgemeinen Schritte zur Absicherung sind bei allen Distributionen gleich. Nur die Art der Umsetzung hängt – wie alles bei K8s – stark von den gewählten Plug-ins ab. Das Beispiel „Policies für Container“ zeigt das gut: Für dieses eine Thema der Absicherung gab es früher die eingebauten Pod Security Policies. Diese sind inzwischen Geschichte und ausgeliefert wird Pod Security Admission mit vordefinierten Sicherheitslevels. Stattdessen lassen sich auch Cloud Workload Protection Platforms (CWPP, [1]) oder Open-Source-Tools wie der Open Policy Agent (OPA) nutzen. Aber wie immer bei K8s gibt es natürlich noch mehr Tools.

Und dann ist da noch die Frage, an welcher Stelle man die Policies anwendet. Das kann in der CI/CD-Pipeline vor dem Build des Container-Images passieren oder in der Registry und auf dem K8s-Master, über einen Admission Controller. Jede K8s-Distribution und jeder Managed Service hat dabei eigene Vorstellungen und implementiert es anders. Besonders die großen Hyperscaler mit ihren sehr umfangreichen Angeboten haben hier teilweise komplett eigene Lösungen im Angebot.

Standards zur Absicherung

Wer es sich hier einfach machen will, nutzt die beiden Bausteine „SYS.1.6 Containerisierung“ und „APP.4.4 Kubernetes“ des BSI-Grundschutz-Kompendiums als Einstieg. Sie enthalten alle Sicherheitsmaßnahmen, die in jedem Cluster und in jeder CI/CD vorhanden sein sollten [2]. Andere Standards sind beispielsweise der Kubernetes-Härtungs-Guide der NSA, der etwas in die Jahre gekommene NIST 800-190 Application Container Security Guide und natürlich der CIS-Benchmark für die Details der Konfiguration. Die OWASP hat auch zu diesem Thema eine Top 10 (siehe ix.de/zprh).

Für den Einstieg empfiehlt der Autor die BSI-Bausteine, die sowohl Technik als auch Organisation betrachten und die er selbst initiiert und geschrieben hat.

Was man immer braucht

Egal, welche Distribution oder welcher Managed Service, die grundlegenden Aspekte der Absicherung sind immer die gleichen und bei den meisten Angeboten bereits abgedeckt. Zumindest ist die

iX-TRACT

- Kubernetes-Umgebungen haben aus der Sicherheitsperspektive gegenüber Legacy-IT einige konzeptionelle Vorteile.
- Kubernetes in Rohform erfordert bei der sicheren Konfiguration einen hohen Aufwand.
- Kubernetes-Distributionen bringen zwar schon viele Sicherheitskomponenten mit, müssen aber auch entsprechend konfiguriert werden.
- Die Container- und Kubernetes-relevanten Bausteine des BSI IT-Grundschutzes bieten eine gute Grundlage zur Systemhärtung.

© Copyright by Heise Medien.
Persönliches PDF für Christoph Puppe aus 99096 Erfurt

Aufbau der Tutorialserie

Dieser Beitrag ist der Auftakt einer dreiteiligen Tutorialserie, die verschiedene Teilbereiche der Kubernetes-Sicherheit abdeckt. Er zeigt die Grundlagen der Absicherung, welche Themen dabei zu beachten sind und wo Härtung notwendig ist. In den nächsten Ausgaben wird das Tutorial dann notwendigen Schritte für die Public-Cloud-Angebote und die On-Premises-Distributionen aufzeigen.

Begleitet wird diese Serie von Wissensartikeln, die spezifische Aspekte beleuchten. In dieser Ausgabe sind dies SECCOMP und wie Kubernetes diese Sicherheitsprofile einbindet. In der nächsten Ausgabe steht die Authentifizierung innerhalb und gegen Kubernetes im Fokus, danach zeigt ein weiterer Beitrag, wie Kubernetes Anfragen verifiziert und kontrolliert, also wie Admission Controller arbeiten.

Möglichkeit zur Absicherung gegeben und die Werkzeuge sind schon vorhanden. Nur wer einen Vanilla-K8s-Cluster beispielsweise mit kubeadm oder kubespray installiert, muss auch selbst die Grundlagen schaffen: Werkzeuge installieren und konfigurieren.

Die Authentifizierung und rollenbasierte Berechtigungen zu aktivieren ist nicht schwer. Hierfür startet man im ersten Schritt per

```
kube-apiserver --authorization-↵
          mode=RBAC --other-options ↵
                        --more-options
```

den API-Server. Danach erkennt er die API rbac.authorization.k8s.io, die vier Objekte vorsieht: `Role`, `ClusterRole`, `RoleBinding` und `ClusterRoleBinding`. Da K8s selbst aber keine Liste von Benutzern und deren Anmeldeinformationen hat, muss man noch einen Dienst vorschalten, der die Anmeldung übernimmt und ein Token erstellt, das der K8s-API-Server akzeptiert und das einem Account zugeordnet ist.

Dieses kleine Beispiel zeigt, dass zu jeder der im Folgenden erwähnten notwendigen Einstellungen einiges an Vorbereitung und ein nicht unerheblicher Aufwand für die Umsetzung gehört. Da bei allen Distributionen und Managed Services die Umsetzung unterschiedlich ist, geht der Artikel hier nur auf das Ziel der Sicherheitsmaßnahme ein und nicht auf die Details der Einstellungen.

OWASP Kubernetes Top Ten 2022

1. unsichere Workload-Konfigurationen
2. Schwachstellen in der Lieferkette
3. übermäßig freizügige RBAC-Konfigurationen
4. fehlende zentralisierte Durchsetzung von Richtlinien
5. unzureichende Protokollierung und Überwachung
6. defekte Authentifizierungsmechanismen
7. fehlende Kontrollen der Netzwerksegmentierung
8. Fehler bei der Verwaltung von Geheimnissen
9. fehlkonfigurierte Clusterkomponenten
10. veraltete und anfällige Kubernetes-Komponenten

Security 101 – Container und Image Policies

Der BSI-Grundschutz sagt in „SYS.1.6.A10 Richtlinie für Images und Container-Betrieb (S)“ und „APP.4.4.A13 Automatisierte Auditierung der Konfiguration (H)“, dass eine Festlegung erfolgen muss, wie Images aussehen und wie Container sich verhalten dürfen. Die Anforderung „SYS.1.6.A6 Verwendung sicherer Images (B)“, die festlegt, dass Images nur aus sicheren Quellen zu beziehen sind und dass das Image keine Verwundbarkeiten enthält, wird von den meisten Anwendern eingehalten. In eigentlich allen Image-Repositorys, seien es öffentliche wie der Docker Hub oder Quay.io oder selbst betriebene wie Harbor, stehen Scanner für Verwundbarkeiten zur Verfügung. Aber wie das BSI es mit dem (B) anzeigt, ist dies eine Basis-Anforderung. Die Vorteile bei der Sicherheit, die mit Containern möglich sind, kommen erst zum Tragen, wenn auch die Standard-Anforderungen (S) und die erhöhte Anforderung APP.4.4.A13 (H) umgesetzt sind.

Diese Anforderungen verlangen vom Betreiber, dass nicht nur eine organisatorische Richtlinie existiert, sondern auch eine technische Umsetzung erfolgt. K8s muss dafür ein zusätzliches Werkzeug, etwa einen Admission Controller, enthalten, das sicherstellt, dass wirklich nur Images starten und Container laufen, die

© Copyright by Heise Medien.
Persönliches PDF für Christoph Puppe aus 99096 Erfurt

Kreuzreferenztabelle des BSI-IT-Grundschutz-Bausteins SYS.1.6 2022

SYS.1.6	Name	CIA[1]	G 0.14[2]	G 0.18[2]	G 0.19[2]	G 0.20[2]	G 0.21[2]	G 0.22[2]
SYS.1.6.A1	Planung des Container-Einsatzes		✓	✓	✓	✓	✓	✓
SYS.1.6.A2	Planung der Verwaltung		✓	✓	✓		✓	✓
SYS.1.6.A3	Sicherer Einsatz containerisierter IT-Systeme		✓		✓	✓	✓	✓
SYS.1.6.A4	Planung der Bereitstellung und Verteilung von Images			✓		✓	✓	✓
SYS.1.6.A5	Separierung der Administrations- und Zugangsnetze bei Containern		✓		✓			✓
SYS.1.6.A6	Verwendung sicherer Images		✓		✓	✓	✓	
SYS.1.6.A7	Persistenz von Protokollierungsdaten der Container		✓					
SYS.1.6.A8	Speicherung von Zugangsdaten bei Containern		✓		✓			
SYS.1.6.A9	Eignung für Container-Betrieb				✓	✓	✓	
SYS.1.6.A10	Richtlinie für Images und Container-Betrieb		✓		✓	✓	✓	
SYS.1.6.A11	Nur ein Dienst pro Container				✓			
SYS.1.6.A12	Verteilung sicherer Images		✓			✓	✓	✓
SYS.1.6.A13	Freigabe von Images					✓	✓	✓
SYS.1.6.A14	Aktualisierung von Images		✓	✓		✓	✓	
SYS.1.6.A15	Limitierung der Ressourcen pro Container							
SYS.1.6.A16	Administrativer Fernzugriff auf Container		✓		✓		✓	
SYS.1.6.A17	Ausführung von Containern ohne Privilegien		✓					
SYS.1.6.A18	Accounts der Anwendungsdienste		✓		✓			
SYS.1.6.A19	Einbinden von Datenspeichern in Container		✓		✓		✓	✓
SYS.1.6.A20	Absicherung von Konfigurationsdaten						✓	✓
SYS.1.6.A21	Erweiterte Sicherheitsrichtlinien	CIA	✓		✓			
SYS.1.6.A22	Vorsorge für Untersuchungen	CI		✓				
SYS.1.6.A23	Unveränderlichkeit der Container	I						✓
SYS.1.6.A24	Hostbasierte Angriffserkennung	CIA	✓	✓	✓		✓	✓
SYS.1.6.A25	Hochverfügbarkeit von containerisierten Anwendungen	A						
SYS.1.6.A26	Weitergehende Isolation und Kapselung von Containern	CI	✓		✓			

[1] vorrangig geschützte Grundwerte: Confidentiality (Vertraulichkeit), Identity (Identität), Availability (Verfügbarkeit), [2] G.xx: elementare Gefährdungen laut IT Grundschutz

die Regeln einhalten. Der Teufel steckt auch hier aber im Detail. Es genügt nicht, den Open Policy Agent (OPA), Kyverno oder eines der diversen kommerziellen Tools zu installieren. Es sind auch die Regeln zu erstellen und zu testen. Listing 1 zeigt als Beispiel eine Regel im OPA, die die Herkunft der Images kontrolliert. Sie erlaubt nur die Registry hooli.com.

Hier haben die kommerziellen Tools den Vorteil, dass sie auch selbstlernend agieren und aus dem Verhalten von Containern in der Testumgebung mehr oder weniger selbstständig Regelsätze erstellen, die dann auf die Container in der Produktion Anwendung finden. Nach über 20 Jahren Erfahrung mit AppArmor, SECCOMP und SELinux ist bekannt, dass die Regelwerke im Alltag kaum zum Einsatz kommen, weil der Aufwand für die Erstellung und das Testen so hoch ist, dass so gut wie niemand die Möglichkeiten, die Anwendungen einzuschränken, wirklich ausnutzt. Hier kann die Automatisierung der Sicherheit und damit auch „Security as Code“ eine erhebliche Verbesserung bringen und Umgebungen sehr viel besser vor Angriffen schützen, als dies in der traditionellen IT möglich war.

Auch können diese Tools die BSI-Anforderung „SYS.1.6.A21 Erweiterte Sicherheitsrichtlinien (H)“ umsetzen, die SECCOMP fordert. Die Anforderung verlangt zwar keine Automatisierung, ist aber in der Praxis ohne automatsierte Umgebungen kaum realisierbar. Details hierzu liefert der Artikel „SECCOMP und SELinux für Container in Kubernetes“.

Security 102 – IAM

Das BSI legt in „APP.4.4.A3 Identitäts- und Berechtigungsmanagement bei Kubernetes (B)“ fest, dass alle APIs der Control Plane generell nur authentifizierte Anfragen bearbeiten dürfen und dass jede Anfrage gegen ein Regelwerk zu autorisieren ist. Auch dürfen Bediener und Dienstprogramme nur minimale Berechtigungen erhalten. Das erscheint zwar wie eine Selbstverständlichkeit, aber wer sein Vanilla K8s auspackt, wird schnell feststellen, dass diese Funktion per Default fehlt. RBAC ist ein eigener Admission Controller, den der Benutzer installieren und aktivieren muss. Alle bekannten Distributionen und Managed Services enthalten dies und bringen auch einen Authentifizierungsproxy mit, der etwa aus Anmeldungen an einer Webseite gegen LDAP die von K8s benötigten Token erstellt und an die Anfragen anhängt. Mehr dazu im nächsten Heft im zweiten Tutorialteil.

Security 103 – Network Policies

Zwischen den Containern spannt K8s ein Software-defined Network auf. Dieses Overlay genannte und vom Container Network Interface (CNI) verwaltete Netz

Listing 1: Beispiel OPA-Regel

```
package kubernetes.admission
deny[msg] {
 input.request.kind.kind == "Pod"
     image := input.request.object.spec.containers[_].image
     not startswith(image, "hooli.com")
     msg := sprintf("image fails to come from trusted registry: %v", [image])
}
```


© Copyright by Heise Medien.
Persönliches PDF für Christoph Puppe aus 99096 Erfurt

G 0.23[2]	G 0.24[2]	G 0.25[2]	G 0.27[2]	G 0.28[2]	G 0.30[2]	G 0.31[2]	G 0.32[2]	G 0.37[2]	G 0.39[2]	G 0.40[2]	G 0.45[2]	G 0.46[2]
✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓
✓	✓	✓	✓	✓	✓	✓	✓					✓
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		✓	✓		✓	✓			✓			
✓		✓	✓	✓	✓	✓	✓		✓			✓
✓		✓	✓		✓	✓	✓	✓	✓	✓		✓
					✓			✓			✓	
✓					✓		✓					✓
	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓
✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		✓										
				✓	✓	✓	✓	✓	✓		✓	✓
		✓	✓		✓			✓	✓		✓	✓
		✓	✓					✓	✓		✓	✓
			✓	✓					✓	✓		
✓		✓		✓	✓	✓	✓		✓			✓
				✓	✓	✓	✓		✓			✓
✓				✓	✓	✓	✓		✓		✓	✓
			✓						✓		✓	✓
								✓				
✓			✓	✓	✓	✓	✓		✓	✓		✓
								✓				
				✓					✓			✓
				✓	✓	✓	✓		✓	✓		✓
✓	✓	✓	✓							✓		
✓				✓	✓				✓			✓

hat in der Grundkonfiguration keine Einschränkungen. Alle Container können sich gegenseitig sehen und erreichen. Aber die Container können über das Routing des Kernels auch die Netzwerkinterfaces der Nodes, also das Underlay-Netzwerk, erreichen. Damit könnte ein Angreifer aus dem Container heraus die APIs aller betriebenen Dienste und auch der Control Plane angreifen. Um das zu verhindern, bringen die meisten CNIs die Option mit, Firewall-Regeln im Overlay einzurichten. Listing 2 zeigt, wie das Open-Source-Container- und Netzwerktool Calico es erlaubt, dass im Namespace `production` alle mit `blue` getaggten Dienste auf Port 6379 von mit `red` getaggten Diensten zugreifen dürfen. Listing 3 beschreibt, wie das Netzwerk-Plug-in Cilium alle Verbindungen für die Container im Namespace `empire` erlaubt.

Das BSI fordert in „SYS.1.6.A5 Separierung der Administrations- und Zugangsnetze bei Containern (B)“ und in „APP.4.4.A7 Separierung der Netze bei Kubernetes (S)“, dass das CNI mindestens die Verbindungen zwischen den Containern der Anwendungen und den Containern und Servern der Control Plane verhindert.

Über diese Basisanforderung hinaus ist es unabdingbar, dass das CNI mindestens zwischen Namespaces die Kommunikation auf das minimal Notwendige einschränkt. Wer dann noch die erhöhte Anforderung „APP.4.4.A18 Verwendung von Mikro-Segmentierung (H)“ umsetzen möchte, benötigt Firewall-Regeln im CNI auch zwischen den Containern eines

Listing 2: Calico-Beispiel

```
apiVersion: projectcalico.org/v3
kind: NetworkPolicy
metadata:
  name: allow-tcp-6379
  namespace: production
spec:
  selector: color == 'red'
  ingress:
  - action: Allow
    protocol: TCP
    source:
      selector: color == 'blue'
    destination:
      ports:
        - 6379
```

Listing 3: Cilium-Beispiel

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "allow-within-namespace"
specs:
  - endpointSelector:
      matchLabels: {}
    egress:
    - toEndpoints:
      - matchLabels:
          "k8s:io.kubernetes.pod.namespace": empire
    ingress:
    - fromEndpoints:
      - matchLabels:
          "k8s:io.kubernetes.pod.namespace": empire
```


© Copyright by Heise Medien.
Persönliches PDF für Christoph Puppe aus 99096 Erfurt

Kreuzreferenztabelle des BSI-IT-Grundschutz-Bausteins APP.4.4 Kubernetes 2022

APP.4.4	Name	CIA [1]	G 0.14 [2]	G 0.19 [2]	G 0.20 [2]	G 0.21 [2]
APP.4.4.A1	Planung der Separierung der Anwendungen		✓	✓		
APP.4.4.A2	Planung der Automatisierung mit CI/CD		✓	✓	✓	✓
APP.4.4.A3	Identitäts- und Berechtigungsmanagement bei Kubernetes		✓	✓		
APP.4.4.A4	Separierung von Pods		✓	✓		✓
APP.4.4.A5	Datensicherung im Cluster					
APP.4.4.A6	Initialisierung von Pods					
APP.4.4.A7	Separierung der Netze bei Kubernetes		✓	✓		✓
APP.4.4.A8	Absicherung von Konfigurationsdateien bei Kubernetes		✓	✓		✓
APP.4.4.A9	Nutzung von Kubernetes Service-Accounts		✓			✓
APP.4.4.A10	Absicherung von Prozessen der Automatisierung		✓	✓		✓
APP.4.4.A11	Überwachung der Container					
APP.4.4.A12	Absicherung der Infrastruktur-Anwendungen		✓	✓		✓
APP.4.4.A13	Automatisierte Auditierung der Konfiguration	CIA	✓	✓		✓
APP.4.4.A14	Verwendung dedizierter Nodes	CIA	✓	✓		
APP.4.4.A15	Trennung von Anwendungen auf Node- und Cluster-Ebene	CIA	✓	✓		
APP.4.4.A16	Verwendung von Operatoren	CIA				
APP.4.4.A17	Attestierung von Nodes	CIA	✓	✓		
APP.4.4.A18	Verwendung von Mikro-Segmentierung	CI	✓	✓		
APP.4.4.A19	Hochverfügbarkeit von Pods	A				
APP.4.4.A20	Verschlüsselte Datenhaltung bei Pods	C	✓	✓		
APP.4.4.A21	Regelmäßiger Neustart von Pods	CIA	✓	✓		

[1] vorrangig geschützte Grundwerte: Confidentiality (Vertraulichkeit), Identity (Identität), Availability (Verfügbarkeit), [2] G.xx: elementare Gefährdungen laut IT Grundschutz

Namespace oder verwendet gleich ein Service Mesh mit mTLS-Authentifizierung wie ISTIO.

Security 104 – Verwundbarkeitsmanagement

Wie schon bei den Policies für Container geschrieben, dürfen Images keine Verwundbarkeiten enthalten. Dies ist in der Praxis nur selten zu 100 Prozent umsetzbar. Die übliche Regelung ist, dass Verwundbarkeiten mit einem CVSS über 8 komplett verboten sind. Falls es doch ein Image gibt, das der Betrieb unbedingt benötigt, kann der Sicherheitsbeauftragte dies für eine kurze Zeit genehmigen. Verwundbarkeiten mit geringeren CVSS muss der Betrieb zumindest auf ihre Relevanz untersuchen, also analysieren, ob die Verwundbarkeit im konkreten Szenario tatsächlich ausnutzbar ist. Sofern dies nicht der Fall ist, kann der Betrieb erlaubt sein. Hier muss jede Organisation einen eigenen Umgang finden, denn durch die etwa via Trivy von AquaSec erfolgte Analyse lässt sich eine sehr viel detailliertere Überwachung der Schwachstellen in den Containern realisieren, als dies in der Legacy-IT jemals möglich war. Die bessere Verwaltung der Schwachstellen bringt zusätzliche Sicherheit und für die Einstufung als erträglich lassen sich härtere Grenzen durchsetzen.

Auch die Nodes, die K8s-Distribution und die diversen Werkzeuge müssen natürlich einem Verwundbarkeitsmanagement unterliegen. Regelmäßige Updates sind einzuplanen und auch die Betriebsumgebung ist so einzurichten, dass diese im laufenden Betrieb erfolgen können.

Security 105 – Secrets und andere Token

Früher galt Verschlüsselung und Authentifizierung mit Shared Secret, also einem beiden Seiten bekannten Geheimnis, als verpönt und wurde möglichst durch asymmetrische Verschlüsselung wie RSA ersetzt. Heute gibt es für alles Token: JWT, OpenID, K8s, alle APIs verwenden diese. Und ein Token ist nichts anderes als ein Shared Secret. Also die schlechteste aller Optionen für die Authentifizierung der Gegenstelle. Diese Token sind nichts anderes als Base64-codierte 256-Bit-Strings. Die Wahrscheinlichkeit, dass jemand ein Token errät, ist daher sehr gering. Aber der Diebstahl von Token ist ein unbeherrschbares Risiko. Jeder, der in den Besitz eines fremden Tokens gelangt, kann dies missbrauchen und damit Aktionen ausführen, die für den API-Server nicht von legitimen zu unterscheiden sind. Google war bis vor Kurzem der einzige Anbieter, der bei Token zumindest Regeln definiert, woher die Anfrage mit diesem Token kommen darf. Azure hat hier mit Conditional Access für Service-Accounts schon nachgezogen, K8s selbst, die Werkzeuge der Control Plane oder auch die Distributionen haben derzeit keine Verteidigung gegen Tokenmissbrauch.

Daher ist die mit Abstand wichtigste Sicherheitsmaßnahme der Tokenschutz. Was für Token gilt, trifft auch für andere Shared Secrets wie Passwörter und die für gute Authentifizierung notwendigen privaten Schlüssel zu. Das Secrets Management ist der „Single Point of Hackability“ und auch in K8s enthalten. Zumindest speichert K8s diese Geheimnisse und gibt sie nur an berechtigte Dienste und Personen heraus. Es speichert sie allerdings unverschlüsselt im etcd. Wer das nicht möchte, muss auf Werkzeuge wie die CWPP oder spezialisierte Tools wie HashiCorp Vault zurückgreifen.

Die Secrets liegen allerdings nicht nur im K8s. Auch die Codeverwaltungen, also meistens Git, müssen sie speichern, weil sie sonst keine automatisierten Deployments über Pipelines in die Cluster schieben können. Hier sind die Hersteller mehr oder weniger gut in der Verwaltung und ein kleiner Fehler beim Berechtigungsmanagement für die Secrets kann den Sourcecode Hunderttausender Anwendungen oder die Daten aller aktuell im Betrieb befindlichen Anwendun-

© Copyright by Heise Medien.
Persönliches PDF für Christoph Puppe aus 99096 Erfurt

	G 0.23 [2]	G 0.25 [2]	G 0.27 [2]	G 0.28 [2]	G 0.30 [2]	G 0.37 [2]	G 0.39 [2]	G 0.45 [2]	G 0.46 [2]
	✓			✓					✓
	✓	✓	✓	✓	✓	✓	✓	✓	✓
	✓				✓	✓			✓
	✓			✓			✓	✓	✓
								✓	
		✓							
	✓			✓			✓	✓	✓
	✓			✓			✓	✓	✓
					✓				
	✓	✓			✓				
		✓	✓						
	✓			✓			✓	✓	✓
	✓				✓		✓	✓	✓
	✓			✓				✓	✓
	✓			✓				✓	✓
		✓							
	✓			✓				✓	✓
	✓			✓				✓	✓
		✓							
	✓			✓				✓	✓
	✓			✓				✓	✓

gen gefährden. Beides kommt täglich vor und die Vorfälle häufen sich. Hier rächt sich, wenn manche Anbieter sagen, dass die Identität der einzige Perimeter ist und die Kunden daher vergessen, noch zusätzliche Perimeter in ihre Architekturen einzubauen. Wer in den öffentlichen SaaS-Angeboten wie GitLab, GitHub, Quay et cetera unterwegs ist und dort seinen Code und seine Secrets lagert, hat leider keine weiteren Möglichkeiten für zusätzliche Perimeter, im Unterschied zu Code-Repositorys in geschlossenen Netzen.

Security 106 – Monitoring und Angriffserkennung

Bei Monitoring denkt jeder zuerst an Tools wie Prometheus und Nagios, diese decken aber nur das Sicherheitsziel Verfügbarkeit ab. Um alle Sicherheitsziele zu erreichen, muss es auch eine Überwachung aller Cluster auf Anzeichen von Angriffen (Indicators of Compromise, IoC) geben. Hier bietet K8s die eigenen Logfiles, die Events und die Logfiles der Container an, aber für die Auswertung bringt kaum eine Distribution die Werkzeuge mit. Die Managed Services haben hier mit den eingebauten Sicherheitszentralen einen erheblichen Vorteil [3].

Wer bereits ein SIEM (Security Information and Event Management, Überwachung der Protokolldateien auf Anzeichen eines Angriffs) oder SOAR (Security Orchestration, Automation and Response, quasi ein SIEM ergänzt um automatisierte Reaktionen) sein Eigen nennt, kann seine Cluster ohne größere Umstände dort anschließen, gute Anbieter haben auch schon die Alert-Definitionen für die Control Plane dabei. Für die Anwendungen ändert sich weniger, da nur die Art des Transports der Logfiles aus K8s anders ist. Zu beachten ist es dennoch und das BSI fordert dies auch in „SYS.1.6.A7 Persistenz von Protokollierungsdaten der Container (B)“.

Fazit

Container in Kubernetes sind bereits durch cgroups von anderen Prozessen isoliert. Innerhalb der virtuellen Netze im Cluster lassen sich sehr granulare Regelwerke realisieren. SECCOMP kann die Einschränkung der Prozesse in den Containern erleichtern und das Verwundbarkeitsmanagement ist in Containern einfacher. Zusammengefasst kann all dies die Sicherheit des Betriebs im Vergleich zur Legacy-IT erheblich erhöhen. Aber das erfordert – wie fast alles bei K8s – einige zusätzliche Werkzeuge und die K8s-Umgebung selbst und auch die Landschaft der Werkzeuge ändert sich ständig. Wer die Vorteile nutzen will, muss hier genau hinschauen und sollte so viel wie möglich automatisieren. (avr@ix.de)

Quellen

[1] Christoph Puppe; Blitzableiter und Regenschirme; Tools zur Absicherung von Containern; iX 4/2019, S. 98

[2] Christoph Puppe; Dritter Anlauf; BSI-Anforderungen für Container und Kubernetes; iX 10/2021, S. 92

[3] Christoph Puppe; Krähennest in der Wolke; Cloud Security Operations Center im Vergleich; iX 7/2021, S. 88

[4] Weitere Infos zu den NIST-, NSA- und OWASP-Dokumenten: ix.de/zprh

CHRISTOPH PUPPE

ist Managing IT Security Architect und Auditteamleiter für ISO 27001 nach Grundschutz bei SVA System Vertrieb Alexander GmbH, Mitautor des Grundschutz-Kompendiums und ehemaliger Penetrationstester.

© Copyright by Heise Medien.
Persönliches PDF für Christoph Puppe aus 99096 Erfurt