Sicherheit in Kubernetes

Überlegungen und Best Practices zum Schutz sensibler Workloads

vmware

Inhalt

Einführung 3

Authentifizierung und Autorisierung für die Steuerungsebene 3

Network Access Control 4

Beschränkung von Pod-zu-Pod-Datenverkehr 4

Netzwerkgrenzkontrollen 4

Einschränkungen 4

Zugriffskontrolle auf Anwendungsebene 4

Sidecars und der Service-Mesh-Ansatz 4

Härten von Knoten und Containern während der Laufzeit 5

Durchsetzen von Isolationsrichtlinien 5

Patch-Management und CI/CD-Bereitstellungs-Pipelines 5

Begrenzung der Fluktuation 6

Container-Builds mit übermäßigen Berechtigungen 6

Secrets-Management 6

Identitäts-Secrets 6

Nicht identitätsbezogene Secrets 6

Vorbehalte gegenüber Kubernetes Secrets 6

Sicherheitsüberwachung und -Auditing 7

Protokollierung 7

Netzwerküberwachung 7

Überwachung von Hostereignissen 7

Der nächste Schritt 7

Sie sollten zumindest die rollenbasierte Zugriffskontrolle (RBAC) in Ihrem Cluster aktivieren. RBAC ist in den meisten neueren Installationsprogrammen standardmäßig aktiviert und bietet ein Framework für die Implementierung des Prinzips der minimalen Zugriffsrechte für Mitarbeiter und Anwendungen, die auf die Kubernetes-API zugreifen.

Einführung

Best Practices für die Kubernetes-Sicherheit entwickeln sich rasch weiter. Viele Sicherheitsprobleme früherer Kubernetes-Versionen sind in neueren Versionen inzwischen standardmäßig behoben. Doch wie bei jedem komplexen System gibt es immer noch Risiken, die Sie verstehen sollten, bevor Sie dem System Produktionsdaten anvertrauen. In diesem White Paper werden die wichtigsten Aspekte zusammengefasst, die Sie beachten müssen, wenn Sie sensible Workloads auf Kubernetes hosten.

Die Themen in diesem White Paper sollen Sie dabei unterstützen, potenzielle Risiken in Ihrem Cluster zu verstehen. Das Risiko in Ihrer Umgebung hängt von Ihrem Bedrohungsmodell und den Arten von Anwendungen ab, die Sie in Ihrem Cluster ausführen. Sie müssen abwägen, wie Sie am besten in Sicherheitskontrollen und -härtung investieren. Dabei sind die Sensibilität Ihrer Daten, der Zeit- und Personalaufwand, den Sie für die Sicherheit aufwenden können, sowie die Compliance-Anforderungen Ihrer Branche wie HIPAA, FISMA und PCI DSS zu berücksichtigen.

Authentifizierung und Autorisierung für die Steuerungsebene

Die Kubernetes-Steuerungsebene besteht aus dem Kubernetes-API-Server, dem Kubelet und anderen Kernkomponenten, die gemeinsam Workloads in Ihrem Cluster planen und ausführen. Der Zugriff auf die Steuerungsebene ist vergleichbar mit dem SSH-Zugang oder dem physischen Zugang zu einem Gerät und muss daher sorgfältig kontrolliert werden. Dank der Flexibilität des Kubernetes-Zugriffskontrollsystems können Sie die Zugriffskontrollen an die Anforderungen Ihrer Umgebung anpassen, doch diese Flexibilität stellt mitunter eine Herausforderung dar.

Sie sollten zumindest die rollenbasierte Zugriffskontrolle (RBAC) in Ihrem Cluster aktivieren. RBAC ist in den meisten neueren Installationsprogrammen standardmäßig aktiviert und bietet ein Framework für die Implementierung des Prinzips der minimalen Zugriffsrechte für Mitarbeiter und Anwendungen, die auf die Kubernetes-API zugreifen.

Um RBAC optimal nutzen zu können, benötigen Sie eine geeignete Konfiguration:

- Führen Sie jede Komponente mit den restriktivsten Berechtigungen aus, mit denen die erwartete Funktionalität gerade noch gewährleistet werden kann. Die meisten Anwendungen in Ihrem Cluster benötigen nur wenig oder gar keinen Zugriff auf die Kubernetes-API. Systemkomponenten wie beispielsweise ein Ingress-Controller oder ein Überwachungssystem benötigen eventuell mehr Zugriffsrechte, können jedoch häufig auf einen reinen Lesezugriff oder den Zugriff innerhalb eines bestimmten Namespace beschränkt werden. Die Kubernetes-API-Audit-Protokolle sind ein nützliches Tool, um herauszufinden, welche APIs eine bestimmte Anwendung verwendet.
- Achten Sie darauf, dass vertrauenswürdige Komponenten nicht als Dreh- und Angelpunkte fungieren, über die Anwender mit weniger Zugriffsrechten ihre Berechtigungen ausweiten können. Dem Kubernetes Dashboard und dem Helm-Tiller-Daemon sollten Sie beispielsweise besondere Aufmerksamkeit schenken. Isolieren Sie diese Komponenten durch Authentifizierung und Autorisierung auf Anwendungsebene oder durch Network Access Control, um unbefugten Zugriff zu verhindern.

Ziehen Sie für Anwender in Erwägung, die Kubernetes-Authentifizierung in Ihr bestehendes Unternehmensidentitätssystem zu integrieren. Kubernetes kann sich standardmäßig mit jedem konformen OpenID Connect-Anbieter, wie GitHub oder Google, authentifizieren. Sie können die Kubernetes-Authentifizierung und -Autorisierung auch mit WebHook-basierten Plug-ins erweitern, um eine benutzerdefinierte Identitätsintegration zu erstellen.

Außerdem müssen Sie den Zugriff auf die Infrastruktur unterhalb von Kubernetes schützen. Dazu kann auch der SSH-Zugang oder die Zugriffskontrolle des Cloud-Anbieters gehören, wie z.B. AWS IAM. Diese Systeme sind eine potenzielle Schwachstelle, da sie das Kubernetes-Zugangskontrollsystem umgehen. Ein Angreifer mit SSH-Zugriff auf den Kubernetes etcd-Host kann beispielsweise böswillig Pod-Definitionen einfügen, um beliebigen Code an anderer Stelle des Clusters auszuführen.

Eine Lösung für das Problem der Zugriffskontrolle auf Netzwerkebene ist eine starke Authentifizierung auf Anwendungsebene, wie z.B. die gegenseitige TLS-Authentifizierung. Eine kryptografische Anwendungsidentität ist leistungsfähig, weil damit die Identität über Netzwerkgrenzen hinweg effizient ausgedrückt werden kann.

Network Access Control

Viele bisherige Anwendungen gehen davon aus, dass der Zugang auf Netzebene eine Autorisierungsstufe impliziert. Auch wenn Ihre Anwendungen eine starke Authentifizierung und Autorisierung auf Anwendungsebene beinhalten, bietet die Zugriffskontrolle auf Netzwerkebene einen umfassenden Schutz. Sie schützt wirksam vor Schwachstellen im Vorfeld der Autorisierung, wie etwa der Heartbleed-Schwachstelle (CVE-2014-0160) in OpenSSL.

Beschränkung von Pod-zu-Pod-Datenverkehr

Kubernetes bietet leistungsstarke Kerndatentypen für das Festlegen von Network Access Control zwischen Pods. Netzwerkrichtlinien in Kubernetes können den eingehenden Datenverkehr zu einem Pod basierend auf dem Namespace und den Labels des Quell-Pod sowie der IP-Adresse für Datenverkehr, der von außerhalb des Clusters stammt, begrenzen. In Kubernetes 1.8 kann die Netzwerkrichtlinie auch den ausgehenden Datenverkehr mit demselben Satz von Selectors begrenzen. Ein guter Ausgangspunkt ist die standardmäßige Ingress-Beschränkung zu Ihrem Anwendungs-Namespace, wie in der Dokumentation zu den Kubernetes-Netzwerkrichtlinien beschrieben.

Die Durchsetzung von Netzwerkrichtlinien hängt von Ihrem CNI-Anbieter (Container Network Interface) ab. Nicht alle CNI-Anbieter führen diese Kontrollen durch. Ohne sie versagt Kubernetes beim Öffnen: Die API akzeptiert zwar Ihre Netzwerkrichtlinien, aber die Richtlinien werden nicht durchgesetzt. Wenn Ihnen Network Access Control wichtig ist, sollten Sie sich für einen Anbieter wie Calico entscheiden, der diese Kontrollen implementiert.

Netzwerkgrenzkontrollen

Je nach Ihrer Umgebung möchten Sie vielleicht zusätzlich zu den von Kubernetes erzwungenen Kontrollen auf Pod-Ebene einige Ingress- und Egress-Kontrollen an der Netzwerkgrenze durchsetzen. Insbesondere falls ein Angreifer eine Ihrer Anwendungen infiziert und Ihre Container-Laufzeitumgebung oder Ihren Kernel ausgenutzt hat, können Sie nicht darauf vertrauen, dass der Knoten Network Access Control durchsetzt.

Einschränkungen

In dynamischen Umgebungen wie Kubernetes bestehen für Network Access Controls einige Einschränkungen. Zu den Schwierigkeiten gehören:

- Verbund von Kubernetes-Netzwerkrichtlinien zwischen mehreren Clustern
- Integration von Kubernetes-Kontrollen auf Netzwerkebene und detaillierten Kontrollen auf Netzwerkebene, die außerhalb des Pod-Networking-Layers ausgedrückt werden (z.B. in AWS-EC2-Sicherheitsgruppen)

Wenn Sie auf diese Probleme stoßen, überlegen Sie, ob Sie eine weniger detaillierte Netzwerkrichtlinie verwenden und sich bei der detaillierten Zugriffskontrolle auf die Anwendungsschicht verlassen können.

Zugriffskontrolle auf Anwendungsebene

Eine Lösung für das Problem der Zugriffskontrolle auf Netzwerkebene ist eine starke Authentifizierung auf Anwendungsebene, wie z.B. die gegenseitige TLS-Authentifizierung. Eine kryptografische Anwendungsidentität ist leistungsfähig, weil damit die Identität über Netzwerkgrenzen hinweg effizient ausgedrückt werden kann.

Sidecars und der Service-Mesh-Ansatz

Selbst wenn Anwendungsidentitäten bereitgestellt werden, kann die Nachrüstung bestehender Anwendungen, um gegenseitige Transport Layer Security (TLS) zu implementieren, frustrierend, zeitaufwändig und fehleranfällig sein. Eine Lösung besteht darin, die L7-Autorisierung nach dem Sidecar-Muster zu implementieren. In diesem Modell hat jede Anwendung einen benachbarten Proxy-Daemon, der eingehende Verbindungen beendet und authentifiziert sowie ausgehende Verbindungen transparent authentifiziert. Kubernetes vereinfacht dies, indem mehrere Container zusammen mit einem gemeinsamen Localhost-Netzwerk für den Pod ausgeführt werden können. Wenn dieses Muster im gesamten Cluster verwendet wird, wird es als Service-Mesh bezeichnet. Istio ist eine der frühesten Implementierungen dieses Ansatzes.

Ein erfolgreiches Muster besteht darin, dass die meisten Benutzer nur über eine Bereitstellungs-Pipeline mit dem Produktions-Cluster interagieren.

Härten von Knoten und Containern während der Laufzeit

Sie sollten die Sicherheit an der Grenze zwischen Container und Host im Blick behalten. Dies ist selbst in Einzelmandantenumgebungen wichtig, da eine Schwachstelle für Remotecodeausführung wie Shellshock (CVE-2014-6271) oder die Ruby YAML-Parsing-Schwachstelle (CVE-2013-0156) Ihren ansonsten vertrauenswürdigen Workload in einen bösartigen Agent verwandeln kann. Ohne angemessene Härtung kann eine einzige durch Remote-Codeausführung bedingte Schwachstelle zu einer Übernahme eines ganzen Knotens oder eines ganzen Clusters eskalieren.

Aktuelle Container-Laufzeiten bieten nicht das stärkst mögliche Sandboxing, Sie können jedoch einige Schritte unternehmen, um das Risiko von Container-Escape-Schwachstellen zu minimieren:

- Segmentieren Sie Ihre Kubernetes-Cluster nach Integritätsgrad. Dies ist eine einfache, jedoch sehr wirksame Methode, um das Gefährdungspotenzial von Schwachstellen, die durch einen Ausbruch aus Containern verursacht werden, zu begrenzen. Beispielsweise können Ihre Entwicklungs- und Testumgebungen in einem anderen Cluster gehostet werden als Ihre Produktionsumgebung.
- Investieren Sie in unkompliziertes Host- und Kernel-Patching. Stellen Sie sicher, dass Sie neue System-Updates testen können (beispielsweise in einer Staging-Umgebung) und dass Ihre Anwendungen ein Rolling Upgrade des Clusters ohne Beeinträchtigung der Anwendungsverfügbarkeit tolerieren.

Kubernetes eignet sich hervorragend für die Orchestrierung dieser Upgrades. Sobald Sie sich darauf verlassen können, dass Kubernetes die Anwendungs-Pods dynamisch ausbalanciert, wird das Patch-Management auf Knotenebene relativ einfach. Sie können ein Rolling Upgrade automatisieren, bei dem jeder Knoten ordnungsgemäß entleert und dann entweder an Ort und Stelle ein Upgrade durchgeführt oder (in einer IaaS-Umgebung) der Knoten durch einen neuen Knoten ersetzt wird. Investitionen in diesem Bereich verbessern auch Ihre allgemeine Resilienz gegenüber Ausfällen auf Knotenebene:

- Führen Sie Anwendungen als normaler Anwender und nicht als Root-Anwender aus. „Root (UID 0)“ in einem Linux-Container ist derselbe Anwender wie „root“ auf dem Knoten. Eine Kombination von Sandboxing-Mechanismen schränkt den Handlungsspielraum des im Container ausgeführten Codes ein. Zukünftige Sicherheitslücken im Linux-Kernel werden jedoch eher von einem Root-Anwender ausgenutzt werden können als von einem nicht privilegierten Anwender.
- Aktivieren und konfigurieren Sie zusätzliche Linux-Sicherheitsmodule wie SELinux und AppArmor. Mit diesen Tools können Sie für bestimmte Container ein restriktiveres Sandboxing erzwingen. Sie sind in vielen Situationen nützlich. Allerdings erfordert das Erstellen und Warten geeigneter Konfigurationen einen gewissen Zeitaufwand. Sie sind unter Umständen nicht für jede Anwendung oder Umgebung geeignet.

Durchsetzen von Isolationsrichtlinien

Pod-Sicherheitsrichtlinien bieten einen richtliniengesteuerten Mechanismus, mit dem Sie durchsetzen können, dass Anwendungen in Ihrem Cluster Container-Sandboxing nur auf genehmigte Weise verwenden. Sie können beispielsweise vorschreiben, dass alle Pods in einem bestimmten Namespace nicht mit Root-Berechtigung ausgeführt werden, keine Host-Dateisysteme mounten und kein Host-Networking verwenden.

Patch-Management und CI/CD-Bereitstellungs-Pipelines

Ein erfolgreiches Muster besteht darin, dass die meisten Benutzer nur über eine Bereitstellungs-Pipeline mit dem Produktions-Cluster interagieren. Diese Pipeline besteht aus einem oder mehreren automatisierten Systemen, die bestimmte Aufgaben übernehmen: Sie fügen beispielsweise Code in Container-Images ein, führen Unit- und Integrationstests aus und führen weitere Validierungsschritte durch wie das Anhalten für eine manuelle Genehmigung. Je nach Ihren Anforderungen können Entwickler weiterhin über direkten Lesezugriff auf die Kubernetes-API verfügen oder bei einem Vorfall in Pods eskalieren.

Eine robuste Pipeline für die Anwendungsbereitstellung ist auch der Schlüssel dazu, Schwachstellen in Container-Images zu beheben. Sie können Tools wie Clair nutzen, um bekannte Schwachstellen in den von Ihnen verwendeten Bibliotheken und Paketen zu identifizieren. Um Patches jedoch zeitnah zu veröffentlichen, benötigen Sie eine zuverlässige, automatisierte Methode, mit der Sie gepatchte Versionen des Containers neu erstellen und testen.

Sicherheitsüberwachung basiert auf Protokollierung. Anwendungsprotokolle, Protokolle auf Hostebene, Kubernetes-API-Audit-Protokolle und Protokolle von Cloud-Anbietern (falls zutreffend) sollten Sie generell überwachen. Es gibt bewährte Muster für die Implementierung der Protokollaggregation in gängigen Cluster-Konfigurationen.

Begrenzung der Fluktuation

Intakte Kubernetes-Cluster sind dynamische Umgebungen. Neue Versionen von Anwendungen werden bereitgestellt, Knoten verschwinden bei einem Kernel-Upgrade, Bereitstellungen skalieren vertikal, und Anwender merken meist nichts davon. Damit all dies in der Praxis funktioniert, ist ein gewisses Maß an Sorgfalt erforderlich. Dies ist jedoch eine Voraussetzung, um alle Vorteile von Kubernetes nutzen zu können.

Ein Tool, mit dem Sie das in Ihrem Cluster entstandene Chaos eindämmen können, ist das Pod Disruption Budget. Es ist nützlich, wenn Sie über mehrere automatisierte Systeme verfügen und sicherstellen möchten, dass diese nur auf die gewünschte Weise interagieren. Ein Fehler auf Anwendungsebene kann beispielsweise dazu führen, dass einige Pods Ihrer Anwendung vorübergehend nicht verfügbar sind. Ein Pod Disruption Budget kann sicherstellen, dass ein automatisches Rolling Node-Upgrade die verbliebenen intakten Kopien Ihrer Anwendung nicht abbricht.

Container-Builds mit übermäßigen Berechtigungen

Ein Docker-spezifisches Anti-Muster, das Sie in Ihrer Build-Pipeline vermeiden sollten, besteht darin, den Docker-Kontrollsocket auf Hostebene */var/run/docker.sock* während eines Builds in einen Container zu mounten. Der Zugriff auf diesen Socket ist gleichbedeutend mit „root“ auf dem Host, was bedeutet, dass jeder ausgeführte Build den Knoten gefährden kann. Dies gilt umso mehr, falls Ihr Build-System Builds vor einer manuellen Codeüberprüfung ausführt, was häufig vorkommt.

Secrets-Management

Kubernetes verfügt über ein zentrales Primitiv für das Management von Anwendungs-Secrets, das entsprechend als Secret oder Geheimnis bezeichnet wird. In der Regel gibt es zwei Hauptgründe, warum Anwendungen Secrets benötigen:

1. Sie benötigen Zugriff auf Anmeldedaten, die ihre Identität gegenüber einem anderen System belegen (z.B. ein Datenbankkennwort oder ein API-Token eines Drittanbieters).
2. Sie benötigen ein kryptografisches Secret für einen bestimmten Ablauf (z.B. einen HMAC-Signaturschlüssel für die Ausgabe von signierten HTTP-Cookies).

Identitäts-Secrets

Für den ersten Anwendungsbereich sollten Sie die Bemühungen des SPIFFE-Projekts (Secure Production Identity Framework For Everyone) und der Container Identity-Arbeitsgruppe um eine langfristige Lösung für die dynamische Bereitstellung eindeutiger Anwendungsidentitäten verfolgen. Kurzfristig gibt es keine etablierte Best Practice in diesem Bereich, aber einige Anwender haben Erfolg bei der Integration in bestehende Workflows für die Bereitstellung vorhandener Zertifikate als Teil einer Pipeline für Continuous Integration (CI)/Continuous Delivery (CD). Einfache Kubernetes-native Lösungen, wie z.B. cert-manager, sind eventuell auch für Ihren Anwendungsbereich geeignet.

Nicht identitätsbezogene Secrets

Best Practices für den zweiten Anwendungsbereich befinden sich noch in der Entwicklung, aber viele Anwender integrieren in Systeme wie Vault, die kryptografische Abläufe in einem zentralisierten Service durchführen. Vergewissern Sie sich, dass Sie die gesamte Nachweiskette verstehen, die mit der Authentifizierung gegenüber einem solchen System verbunden ist. Häufig werden dabei als ein Schritt in der Kette Kubernetes Secret-Ressourcen benötigt. Das Back-End für die Vault Kubernetes-Autorisierung authentifiziert beispielsweise Pods, indem es ein Token für ein Kubernetes-Servicekonto nutzt. Dieses Token wird jedoch als Secret-Objekt gespeichert, bevor es in den Pod injiziert wird. Bei diesem Muster müssen Sie außerdem darauf vertrauen, dass Vault Ihr Token nicht wiedergibt und sich nicht gegenüber der Kubernetes-API als Pod ausgibt.

Vorbehalte gegenüber Kubernetes Secrets

Leider bestehen einige Vorbehalte gegenüber Kubernetes-Secrets. Der wichtigste Vorbehalt ist, dass viele häufig verwendete Komponenten, wie beispielsweise Ingress-Controller, derzeit Leseberechtigung für alle Secrets in Ihrem Cluster benötigen. Secrets werden auch nicht standardmäßig im Ruhezustand verschlüsselt. Alpha-Unterstützung für Verschlüsselung ist zwar in Kubernetes 1.7 verfügbar, wird jedoch noch nicht für den Einsatz in der Produktion empfohlen. Eine Abhilfemaßnahme ist die Verwendung von Verschlüsselung auf Volume-Ebene für Ihre etcd-Daten-Volumes (z.B. über dm-crypt oder Volume-Verschlüsselung von Cloud-Anbietern).

Wie Sie Ihren Kubernetes-Cluster schützen, hängt zum Teil von den verfügbaren Ressourcen und den Anwendungsanforderungen ab. Betrachten Sie jedes Element im gesamten Sicherheitskontext und nehmen Sie sich Zeit, um im Vorfeld zu beurteilen, wie wichtig es für Ihre allgemeinen Anforderungen ist.

WEITERE INFORMATIONEN ZU CLOUDNATIVER TECHNOLOGIE VON VMWARE

Wenn Sie mehr darüber erfahren möchten, wie VMware Sie dabei unterstützt, cloudnative Anwendungen zu erstellen, auszuführen und zu verwalten, besuchen Sie *https://cloud.vmware.com/*.

Sicherheitsüberwachung und -Auditing

Der richtige Umfang und die richtige Art der Sicherheitsüberwachung für Ihren Cluster hängen weitgehend davon ab, wie viel Zeit und Personal Ihnen zur Verfügung steht, um auf Benachrichtigungen zu reagieren und die Situation im Auge zu behalten. Generell sollten Sie keine Zeit in den Aufbau von Überwachungssystemen investieren, für deren Wartung und Optimierung Ihnen Zeit fehlt. Beginnen Sie mit Workflows in Echtzeit (basierend auf Benachrichtigungen) und sich regelmäßig wiederholenden Workflows (Audit-Reviews) für Analysten oder Betreiber und bauen Sie die für diese Workflows erforderliche Überwachungsplattform auf.

Protokollierung

Sicherheitsüberwachung basiert auf Protokollierung. Anwendungsprotokolle, Protokolle auf Hostebene, Kubernetes-API-Audit-Protokolle und Protokolle von Cloud-Anbietern (falls zutreffend) sollten Sie generell überwachen. Es gibt bewährte Muster für die Implementierung der Protokollaggregation in gängigen Cluster-Konfigurationen.

Zum Zwecke von Sicherheits-Audits sollten Sie in Erwägung ziehen, Ihre Protokolle an einen externen Speicherort zu streamen, auf den Sie von Ihrem Cluster aus mit der Berechtigung „Append-only“ (nur Anfügen) zugreifen können. Bei AWS können Sie beispielsweise einen S3-Bucket in einem isolierten AWS-Konto erstellen und Ihrem Cluster-Protokollaggregator die Zugriffsberechtigung „Append-only“ zuweisen. Auf diese Weise wird sichergestellt, dass Ihre Protokolle selbst im Falle einer umfassenden Kontaminierung des Clusters nicht manipuliert werden können.

Netzwerküberwachung

Netzwerkbasierte Tools zur Sicherheitsüberwachung, wie z.B. ein System zur Erkennung von Eindringversuchen (IDS, Intrusion Detection System) im Netzwerk und Firewalls für Webanwendungen, sind zwar fast sofort einsatzbereit, aber damit sie gut funktionieren, bedarf es einiger Anstrengungen. Das Hauptproblem besteht darin, dass viele Tools davon ausgehen, dass IP-Adressen nützlichen Kontext für Ereignisse bieten. Um diese Tools in Kubernetes zu integrieren, sollten Sie die erfassten Ereignisse mit Namespace-, Pod Name- und Pod Label-Metadaten von Kubernetes anreichern. Dies fügt dem Ereignis einen wertvollen Kontext hinzu, den Sie für Benachrichtigungen oder manuelle Prüfung nutzen können. Dadurch werden diese herkömmlichen Tools in Ihrem Cluster noch leistungsfähiger als in einer herkömmlichen Umgebung. Einige Überwachungstools sind bereits darauf ausgelegt, Kubernetes-Metadaten zu erfassen. Sie können jedoch auch angepassten Code zur Ereignisanreicherung schreiben, um diese Art der Metadatenintegration Tools hinzuzufügen, die nicht auf die Metadatenerfassung ausgelegt sind.

Überwachung von Hostereignissen

Sie können auch ein hostbasiertes IDS, wie etwa die Überwachung der Dateiintegrität und die Protokollierung von Linux-Systemaufrufen (z.B. auditd), direkt mit Kubernetes ausführen. Die Ergebnisse sind jedoch schwer zu verwalten, da der auf einem bestimmten Knoten ausgeführte Workload während der Anwendungsbereitstellung und der Orchestrierung von Pods durch Kubernetes stündlich schwankt.

Um hostbasierte Ereignisse sinnvoll zu nutzen, sollten Sie Ihre bestehenden Tools so erweitern, dass sie Metadaten von Kubernetes Pods oder -Containern in den Kontext der erfassten Ereignisse aufnehmen. Neuere Systeme wie Sysdig Falco enthalten standardmäßig diesen Kontext.

Der nächste Schritt

Wie Sie Ihren Kubernetes-Cluster schützen, hängt zum Teil von den verfügbaren Ressourcen und den Anwendungsanforderungen ab. Betrachten Sie jedes Element im gesamten Sicherheitskontext und nehmen Sie sich Zeit, um im Vorfeld zu beurteilen, wie wichtig es für Ihre allgemeinen Anforderungen ist. Auf einem sehr hohen Niveau passen einige unserer Empfehlungen gut zu größeren Best Practices bei der Bereitstellung. Hier einige Beispiele:

- Automatisierte Bereitstellungspipeline und automatisierter Scheduler: Damit können Sie das Management von Host- und Anwendungs-Patches durch Rolling Upgrades vereinfachen, die in Ihren gesamten Entwicklungszyklus integriert sind.
- Integrierte Zugriffskontrollen auf geeigneten Ebenen.
- Integrierte Protokollierung und Überwachung: Sie protokollieren und überwachen die Performance und Zuverlässigkeit. Zusätzlicher Support für sicherheitsspezifische Ereignisse und Pod-Metadaten ist nicht trivial, aber unerlässlich. Was genau überwacht wird, hängt von Ihren spezifischen Anforderungen ab.

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com Zweigniederlassung Deutschland Willy-Brandt-Platz 2 81829 München Tel.: +49 89 370 617 000 Fax: +49 89 370 617 333 www.vmware.com/de

Artikelnr.: ModApps_DE_T2Sec_Best_Practices_Kubernetes_Security_wp 3/19