Sicherer Betrieb von Containern

Jon-Steven Streller
Fakultät IV - Wirtschaft und Informatik
Hochschule Hannover, Deutschland
jon-steven.streller@stud.hs-hannover.de

I. Einführung

A. Motivation und Hintergrund

Die Möglichkeit, Container zur Bereitstellung von Anwendungen zu verwenden, hat in den letzten Jahren stark zugenommen. [1] Dieses Wachstum kann der effizienten Fähigkeit zugeschrieben werden, Anwendungen in einer isolierten Umgebung auszuführen, Anwendungen schnell und dynamisch zu reproduzieren, und schlussendlich der Vorteil, dass der Hardware-Ressourcenverbrauch bei Anwendungen, die als Container bereitgestellt werden, im Vergleich zu anderen Bereitstellungsmethoden *(zum Beispiel Virtualisierte Maschinen)* deutlich geringer ist.

Der Betrieb von Containern birgt jedoch auch Gefahren. Insbesondere eine fehlende oder fehlerhafte Konfiguration von Containern kann unerwünschte Sicherheitslücken hervorrufen, die zur Kompromittierung von Containern oder im schlimmsten Fall des gesamten System führen kann.

Aus diesem Grund ist es wichtig, sich um den sicheren Betrieb von Containern zu kümmern, um Risiken zu reduzieren und die Sicherheit der verwendeten Images und Container gewährleisten zu können. In dieser Arbeit werden verschiedene Aspekte im Zusammenhang mit dem sicheren Betrieb des Containers untersucht, darunter die Verwendung von vertrauenswürdigen Images, die sichere Konfiguration der Container und die richtige Wahl des Betriebssystems.

B. Zielsetzung der Arbeit

Das Ziel dieser Arbeit ist es herauszufinden, welche potentiellen Sicherheitsrisiken beim Einsatz einer Container-Technologie bestehen und wie durch verschiedene Maßnahmen die Sicherheit des Betriebs von Containerumgebungen erhöht werden kann.

C. Methodik

Um die Risiken und die entsprechenden Maßnahmen für den sicheren Betrieb von Containern zu ermitteln, wurde Literatur, Dokumentation von verschiedenen Betreibern von Container-Technologien (zum Beispiel Docker und Kubernetes) sowie von Betriebssystemherstellern wie Red Hat entsprechende Artikel oder Kapitel, die sich auf den sicheren Betrieb beziehen, herangezogen und evaluiert.

II. Grundlagen

A. Virtualisierung

Virtualisierung ist eine Technologie, die es ermöglicht, mehrere virtuelle Umgebungen auf derselben Hardware auszuführen. Es existieren unterschiedliche Methoden von Virtualisierung. In den nachfolgenden Absätzen werden drei Methoden vorgestellt und die Vor- und Nachteile gegenübergestellt.

1) Bare Metal: „Bare Metal“ *(deutsch: Blankes Metall)* ist streng genommen keine Virtualisierung und wird nur der Vollständigkeit halber aufgeführt.

Der Begriff „Bare Metal“ bezeichnet die reine Hardware, also den physischen Server. Dies bedeutet, dass physische Hardwarekomponenten bereitgestellt werden und nur für diesen Zweck bestimmt sind. Daher wird das gewünschte Betriebssystem direkt auf der Hardware installiert. Es gibt keine andere Abstraktionsebene zwischen der Hardware und dem Betriebssystem.

Ein Vorteil ist, dass die Hardware dediziert ist und daher nicht mit anderen geteilt wird.

Es gibt keine Isolierung zwischen den Anwendungen App 1 bis App 3 *(siehe Abbildung 1)*. Daher können sich laufende Anwendungen versehentlich gegenseitig stören und dadurch den Betrieb beeinträchtigen.

In einem Bare-Metal-System muss die Rechenleistung im Voraus sorgfältig geplant werden. Eine Erweiterung der Rechenleistung ist nur begrenzt möglich und der entsprechende Systemausfall nicht zu vermeiden.

2) Virtualisierte Maschinen: Bei virtuellen *(auch virtualisierte)* Maschinen wird Hardware angeschafft und das entsprechende Betriebssystem installiert, ähnlich wie bei „Bare Metal“. Als zusätzlicher Schritt wird ein Hypervisor installiert. Dies kann beispielsweise mit Microsofts Virtualisierungstechnologie Hyper-V oder unter anderem VirtualBox erfolgen.

Der Hypervisor bietet die Möglichkeit, vorhandene Hardware-Ressourcen durch Abstraktion trennen zu können. Daher ist es möglich, zusätzliche Betriebssysteme (virtuelle Maschine) auf dem Host-Betriebssystem zu installieren und diese unabhängig und parallel zu betreiben, ohne die anderen virtuellen Maschinen zu kennen. Die Hardware-Ressourcen können jeder virtuellen Maschine individuell zugewiesen und auch nachträglich einfach angepasst werden. [2]

Virtuelle Maschinen können über den Hypervisor auf die zugrunde liegende Hardware zugreifen. Natürlich ist es nicht

möglich, einer virtuellen Maschine mehr Ressourcen zuzuweisen, als das Host-System besitzt.

Wenn das Betriebssystem von App 1 *(siehe Abbildung 1)* kompromittiert ist, kann sich der Angreifer nur in der virtuellen Maschine bewegen, auf der App 1 ausgeführt wird. Außerdem kann der Angreifer nicht sehen, ob oder wie viele andere virtuelle Maschinen laufen. Daher weiß der Angreifer nichts über App 2 und App 3 und deren Betriebssysteme.

Zu den Nachteilen dieses Ansatzes gehört ein höherer Verbrauch an Hardware-Ressourcen. Jede bereitgestellte Anwendung hat ihr eigenes Betriebssystem und benötigt wie das Host-Betriebssystem Arbeitsspeicher, Festplattenspeicher und CPU-Leistung. Auch die Wartung ist zeitaufwändiger. Während „Bare Metal“ lediglich ein Betriebssystem hat, welches gewartet werden muss, muss bei dieser Virtualisierungart das Host-Betriebssystem sowie alle virtuellen Maschinen gewartet werden, um einen sicheren Betrieb gewährleisten zu können. [2]

Zu beachten ist, dass es zwei Arten von Hypervisoren gibt. Ein Typ-1-Hypervisor, der direkt auf der Hardware installiert wird, und Typ-2-Hypervisor, der auf dem Betriebssystem ausgeführt wird, wie in der oben beschriebenen Erläuterung. [2]

3) Containerisierung: Im Fokus dieser Arbeit steht die Virtualisierungsart, bei der anstelle eines Hypervisors eine Container-Technologie wie zum Beispiel **Docker** oder **Kubernetes** eingesetzt wird. Mithilfe dieser Engine können unter anderem Container-Images heruntergeladen werden, die ähnlich wie virtuelle Maschinen komplett isoliert von anderen laufenden Containern gestartet werden können. [2] Im Vergleich zu virtuellen Maschinen sind Container-Images jedoch deutlich platzsparender, sodass bereits mit 17 Megabyte ein Apache-Webserver betrieben werden kann, während eine virtuelle Maschine mit einem entsprechenden Betriebssystem schnell mehrere Gigabyte benötigt. [3]

Wie die Container-Engine es genau schafft, dass Container unabhängig und isoliert voneinander laufen können, und trotzdem so Speichereffizient im Vergleich zur einer virtuellen Maschine sind, wird in Linux-Namespace und Container-Technologie näher erläutert.

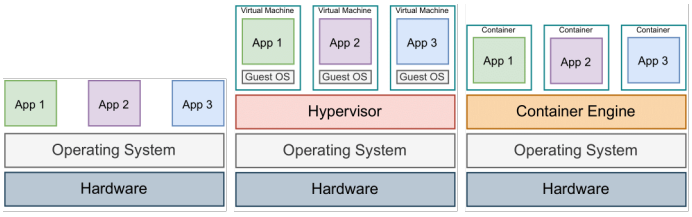


Abbildung 1. Vergleich zwischen Bare-Metal (links), virtuellen Maschinen mit einem Hypervisor Typ-2 (mitte) und eine installierte Container Engine auf dem OS (rechts)

B. Linux-Namespace

Linux-Namespaces sind seit der Kernel-Version 2.4.19 aus dem Jahr 2002 Teil des Linux-Kernels. Durch Linux-Namespaces können verschiedene Arten von Ressourcen, darunter Prozesse (PID), Cgroups, Hardwareressourcen einschließlich CPU und RAM, verwendet und in einer isolierten Umgebung betrieben werden. [4]

Eine weitere Definition lässt sich in der (Linux) Manpage-Dokumentationsbeschreibung für Linux-Namespaces finden:

> A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes. One use of namespaces is to implement containers. [5]

Insbesondere Implementierungen von Containertechnologien wie Docker oder Kubernetes verwenden die Linux-Namespace-Funktion, um jeden einzelnen Container zu isolieren. Die notwendigen Schritte im Hintergrund werden jedoch von der hier eingesetzten Container-Technologie übernommen und durchgeführt, sodass sich als Entwickler keine Gedanken über die korrekte Implementierung der Namespace-Funktionen von Linux gemacht werden müssen. [4]

Es sollte beachtet werden, dass alle Namespace-Instanzen denselben Host-Kernel verwenden. Wenn der Linux-Kernel kompromittiert ist, kann ein Angreifer Zugriff auf alle darauf laufenden Container erlangen.

C. Container-Technologie

In den folgenden Abschnitten werden die einzelnen Komponenten der Container-Technologie und ihre Zusammenhänge prägnant erläutert.

1) Open Container Initiative: OCI ist ein im Jahr 2015 gegründetes Projekt der Linux Foundation. Zu den Mitgliedern zählen Google, IBM, Red Hat, Microsoft und Amazon Web Services (AWS). Ziel ist die Vereinheitlichung und Definition eines Standards für die Containervirtualisierung. [4]

Darüber hinaus veröffentlichte OCI das CLI-Tool „runc“, das für den Betrieb von Containern nach dem OCI-Standard entwickelt wurde. „runc“ ist in vielen gängigen Container-Runtimes implementiert.

2) Container Runtime Interface: Das Container Runtime Interface (CRI) ist ein Standard, der unter anderem definiert, wie die Container-Engine und die Container-Runtime miteinander kommuniziert und wie einzelne Container gestartet, gestoppt, überwacht und Images heruntergeladen beziehungsweise gespeichert werden müssen. [4]

Durch die Implementierung des Standards, ist es der Container-Engine möglich verschiedene Container-Runtimes nahtlos zu unterstützen, was eine höhere Flexibilität zur Folge hat. [4]

Container-Runtimes die diesen Standard implementiert haben sind unter anderem „CRI-O“ und „containerd“.

3) Container-Engine: Die Container-Engine ist die Verbindung zwischen dem Anwender, der zum Beispiel Befehle über die Kommandozeile ausführt, und der Container-Runtime. Darüber hinaus unterstützt die Container-Engine das Container-Management. Dazu gehören unter anderem die Erstellung und Speicherung von Containern, sowie die Orchestrierung wie Skalierung, Verteilung wie Lastverteilung (Load Balancing) und automatisches Up-/ und Down-Skalierung.

Container-Engines sind unter anderem „Kubernetes“, „Docker (Swarm)“ oder auch „Podman“.

4) Container-Runtime: Die Container-Runtime ist für die isolierte und sichere Ausführung des einzelnen Containers auf dem Host-Systems verantwortlich. Um den Betrieb sicherzustellen, bietet die Container-Runtime verschiedene Funktionalitäten, um Container vom Host-System und von anderen Containern zu isolieren. Darüber hinaus überwacht die Container-Runtime die verwendeten Hardware-Ressourcen und schränkt diese bei Bedarf ein, sofern dies konfiguriert wurde. [4]

Container-Runtimes sind unter anderem „CRI-O“, welches von Kubernetes eingesetzt wird, sowie „containerd“, welches bei Docker anzutreffen ist und „runc“, welches von „CRI-O“ und „containerd“ implementiert wird. [4]

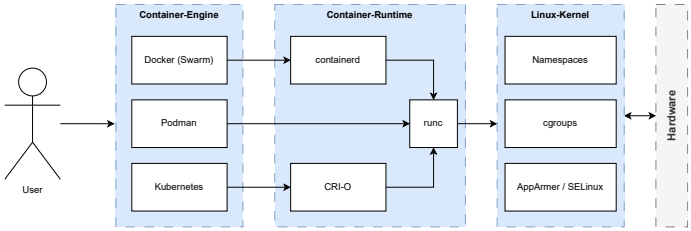


Abbildung 2. In der Abbildung ist das Zusammenspiel zwischen dem Anwender und der Container-Engine (beispielsweise über CLI) sowie der Container-Runtime mit dem Linux-Kernel beschrieben.

5) Image und Container: Ein Image ist ein Paket, das eine vollständige, ausführbare Anwendung bereitstellt, einschließlich aller für die Bereitstellung erforderlichen Abhängigkeiten, Bibliotheken und Anwendungen. Um ein Image zu erstellen, müssen Anweisungen wie zum Beispiel eine Erzeugung einer Datei oder verschiedene Kopiervorgänge, in einer Datei, zum Beispiel Dockerfile, definiert werden. Diese Datei kann dann verwendet werden, um daraus ein Image zu erzeugen. Dieser Vorgang wird auch als „Build Image“ bezeichnet. Im Anschluss erhält man ein Image, welches als statische Vorlage dient und als Grundlage für die Erstellung und Ausführung eines Containers. [6]

Da das Image bereits alle Abhängigkeiten enthält, müssen keine Abhängigkeiten auf dem Host-System installiert werden, die für die Funktion der Anwendung erforderlich sind. Als Administrator ist es somit deutlich einfacher das Host-System in einem sterilen Zustand zu halten.

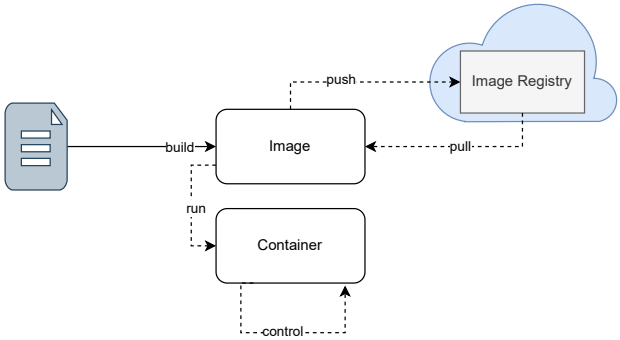


Abbildung 3. Die Anweisungen dazu werden in die Datei geschrieben (für Docker: docker build –file ⟨ Pfad ⟩). Anschließend werden die Anweisungen ausgeführt und ein Image daraus erstellt, das als Grundlage für den Start einer Containerinstanz verwendet werden kann.

D. Common Vulnerabilities and Exposures

Bei „Common Vulnerabilities and Exposures“ (CVE) handelt es sich um eine öffentliche Datenbank die bekannte Schwachstellen in Anwendungen und Hardware enthält. Jeder Eintrag erhält in der Datenbank eine eindeutige Kennung, die als CVE-Nummer (Beispiel: CVE-2023-1255) bezeichnet wird. Diese CVE-Nummer dient zur eindeutigen Identifizierung einer bestimmten Schwachstelle. [7]

Die Vergabe der CVE-Nummern wird von der gemeinnützigen Organisation „MITRE Corporation“ vergeben. Sollte man eine Sicherheitslücke in einer Anwendung finden, kann diese an „MITRE Corporation“ gemeldet *(zum Beispiel über https://cveform.mitre.org)* werden, die im Anschluss diese überprüfen, dokumentieren und mit einer eindeutigen CVE-Nummer in der CVE-Datenbank veröffentlichen. [7]

Ein CVE-Eintrag enthält Informationen über die konkrete Sicherheitslücke, Beschreibung und eine Bewertung der Schwere der Schwachstelle und gegebenenfalls Informationen, wie diese Sicherheitslücke ausgenutzt werden kann. Sollte es bereits einen Patch für die Sicherheitslücke geben, so ist dies ebenfalls direkt im CVE-Eintrag sichtbar *(siehe Abbildung 6)*. [7]

Eine CVE-Datenbank ist ein essentielles Instrument auf Schwachstellen aufmerksam zu machen und das eigene System auf Schwachstellen testen zu können. [7]

III. Potentielle Risiken durch den Betrieb von Containern

In diesem Kapitel werden verschiedene Arten von Risiken durch den Betrieb von Containern und die möglichen Folgen vorgestellt und erläutert. Die Maßnahmen zur Erhöhung der Sicherheit, die die verschiedenen Risiken aufgreifen, werden im darauffolgenden Kapitel (IV) behandelt.

A. Nicht-vertrauenswürdige Image(s)

Jeder Nutzer hat die Möglichkeit Images zu erstellen und diese auch anschließend auf Plattformen wie zum Beispiel Docker Hub (https://hub.docker.com/) zu publizieren.

Genau aus diesem Grund ist es wichtig die Herkunft des Images vor der Inbetriebnahme genaustens zu evaluieren. Images, bei denen keine vertrauenswürdige Basis geschaffen werden konnte, stellen ein erhöhtes Sicherheitsrisiko dar, da diese potenziell unsicheren oder im schlimmsten Fall Schadcode enthalten können. Daraus folgt, dass ein Image beispielsweise mit Malware, Backdoors oder ähnlichen Arten von Viren infiziert sein kann. [8]

Sollte so ein infiziertes Image in einer Containerumgebung ausgeführt werden, kann dies zur weiteren Infektion von Container-Instanzen oder gar des ganzen Host-Systems führen. Auch besteht die Möglichkeit, dass das Image Programmcode eingebettet hat, welches sensible Daten aus dem Container abfängt und diese vertrauliche Information an dritte Server weiterleitet. [8]

Abschließend können nicht-vertrauenswürdige Images unnötige Installationen von Softwarepaketen oder Bibliotheken enthalten, die für den eigentlichen Anwendungsfall unnötig sind, was zur Folge hat, dass die Container-Instanz mehr Hardware verbraucht als nötig und anfälliger auf potenzielle Sicherheitslücken ist, da die entbehrbare Software beziehungsweise Bibliotheken auch auf einem aktuellen Stand gehalten werden muss. [8]

B. Sicherheitslücken im Container

Sicherheitslücken im Container können verschiedene Ursachen haben. Eine häufige Ursache kann durch veraltete Abhängigkeiten, also zum Beispiel veraltete Software oder Bibliotheken, entstehen.

Eine weitere Ursache kann eine mangelnde Isolation von den verschiedenen Containern sein. Sollte sich der Angreifer auf einen Container Zugriff verschafft haben, ist es bei einer mangelnden Isolation einfacher auf andere laufende Container-Instanzen zugreifen zu können.

Auch bei Containern ist besonders drauf zu achten, dass für die verwendeten Container (zum Beispiel MySQL) keine schwachen Passwörter gewählt werden und diese in der kompletten Umgebung einzigartig sind.

Abschließend kann eine fehlende Überwachung und Protokollierung von Ereignissen dazu führen, dass verdächtige Aktivitäten, wie zum Beispiel ein unautorisierter Zugriffsversuch, unentdeckt bleiben. Durch die fehlende Protokollierung ist eine Analyse der Ereignisse schwer vorzunehmen und Angriffe nur bedingt nachvollziehbar.

C. Unzureichende Konfiguration

Die Konfiguration der Container-Technologie ist standardmäßig bereits sicher. In diesem Abschnitt liegt der Fokus auf die Konfiguration der Container an. In diesem Kontext gibt es einige Fälle die Sicherheit beeinträchtigen können.

Geöffnete Ports ohne aktiven Dienst:
Hin und wieder kann es vorkommen, dass bei der initialen Konfiguration einer Orchestrierungsdatei (zum Beispiel docker-compose), auf Anleitungen aus dem Internet zurückgegriffen wird, die fertige Orchestrierungsdateien bereitstellen. Das Problem hierbei ist, dass diese oft unzureichend dokumentiert sind und nicht nachvollzogen, kann, ob jede Portfreigabe auch wirklich nötig ist. Container-Technologien prüfen nicht auf offene Ports. So kann es passieren, dass Ports ohne aktiven Dienst geöffnet sind, was allgemein als schlechte Praxis gilt und das Angriffsrisiko erhöht was wiederum zu unerwünschten sicherheitskritischen Vorfällen führt.

Unzureichende Netzwerkkonfiguration:
Es existieren verschiedene Netzwerkmodi, um Container miteinander kommunizieren zu lassen. Dabei ist der „Host"-Netzwerkmodus besonders kritisch zu betrachten. In diesem Modus verwenden die Container das gleiche Netzwerk wie das Host-System, anstatt eine eigene, isolierte Netzwerkinterface zu besitzen.

In vereinzelten Fällen kann dies eine gute Wahl sein. Sobald allerdings mehrere Container in diesem Modus laufen, die vom Anwendungsfall nichts miteinander zu tun haben sollen, wird an dieser Stelle die Isolation geschwächt, da die Container untereinander erreichbar sind und somit auch miteinander kommunizieren können, was ein Sicherheitsrisiko darstellen kann. Ein Angreifer könnte auf einem kompromittierten Container durch Port-/ beziehungsweise Netzwerkscans die anderen Container entdecken und Angriffmaßnahmen vornehmen.

Unzureichende Beschränkung des Hardware-Ressourcenverbrauchs:
Container-Technologien besitzen die Funktion jedem Container individuell Ressourcen zuzuweisen und zu überwachen.

Sollte die Implementierung von solchen Begrenzungen, wie zum Beispiel CPU-Begrenzungen oder Speicherbeschränkungen nicht korrekt umgesetzt werden, kann es zu Konflikten innerhalb der laufenden Container kommen. Ein einzelner Container kann alle verfügbaren Hardware-Ressourcen verwenden und dadurch die Leistung und Verfügbarkeit von anderen laufenden Container oder des Host-Systems beeinträchtigen.

Sollten die laufenden Container übermäßig Hardware-Ressourcen verwenden, kann die Stabilität des gesamten Host-Systems darunter leiden. Daraus resultiert, dass das Host-System oder andere Container aufgrund der ausgelasteten Hardware abstürzen oder ein unvorhergesehenes Verhalten aufweisen.

Sollte das Host-System auf einer Cloud-basierten Plattform ausgeführt sein, kann eine unzureichende Beschränkung der Hardware-Ressourcen zu übermäßigen Kosten führen.

D. Unzureichendes Patchmanagement

Ein Patchmanagement ist für den sicheren Betrieb von Containern essentiell. Zu beachten ist, dass das Host-Betriebssystem und die laufenden Container-Instanzen im

Patchmanagement vorhanden sein müssen.

Durch ein unzureichendes Patchmanagement altert das Betriebssystem und die Softwarekomponenten im Image, was wiederum zu potenziellen Sicherheitslücken führt und diese können ausgenutzt werden, um einen unautorisierten Zugriff auf das Host-Betriebssystem oder in eine laufende Container-Instanz zu erlangen. Ein daraus resultierendes Problem ist ein potenzieller Datenverlust. Der unautorisierte Zugriff könnte auf vertrauliche Daten zugreifen und, falls die Berechtigungen des komprimierten Benutzers dies zulassen, unwiderruflich löschen.

Auch wenn es nicht immer zu einer Infiltrierung des Systems oder Container kommen muss, so kann ein unzureichendes Patchmanagement zumindest erhöhte Leistungseinbußen oder gar Ausfallzeiten verursachen, da möglicherweise wichtige Optimierungen oder Fehlerbehebungen fehlen.

Insbesondere ist darauf zu achten, dass jeder laufende Container gepatcht werden muss. Sollte lediglich nur ein Container unzureichend gepatcht sein, so kann dies unmittelbar Auswirkungen auf die anderen laufenden Container-Instanzen haben und eine Vielzahl von Problemen auslösen.

IV. Massnahmen zur Erhöhung der Sicherheit

In diesem Kapitel werden gängige Maßnahmen zur Erhöhung der Sicherheit des Containerbetriebs vorgestellt. Die Darstellung gliedert sich in drei Unterkapitel, die jeweils unterschiedliche Abstraktionsebenen (Absicherung der Host-Umgebung, Absicherung des Images und Absicherung des Containers) und entsprechende Maßnahmen auf der Abstraktionsebene beinhalten.

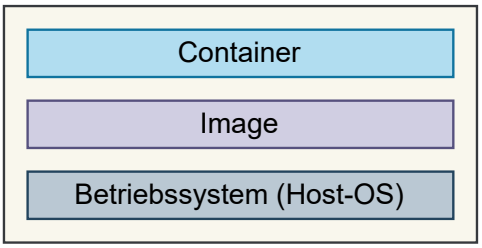


Abbildung 4. Die drei Abstraktionsebenen die in diesem Kapitel betrachtet werden

A. Absicherung der Host-Umgebung durch Container Operating System

Ein Container Operating System ist ein Betriebssystem, das speziell für die Ausführung von Container-Engines konzipiert und entwickelt wurde. Im Gegensatz zu normalen Betriebssystemen werden nur die Anwendungen installiert, die für die ordnungsgemäße Ausführung der Container-Engine erforderlich sind. Daher ist das Ziel, eine möglichst kleine Patchfläche zu schaffen, was auch zu weniger potenziellen Sicherheitslücken führt. Dank des Verzichts von entbehrlichen Anwendungen kann ein Container Operating System schnell gestartet und bereitgestellt werden im Vergleich zu einem vollumfänglichen Betriebssystem.

Darüber hinaus bieten Container Operating Systems unterschiedliche Konzepte, die unabhängig voneinander konfiguriert werden können, um die Sicherheit zu erhöhen. Diese werden in den folgenden Absätzen ausführlich erläutert.

Unveränderliches *(Immutable)* Dateisystem:
Container Operating Systems bieten oft die Möglichkeit, das Betriebssystem in einen unveränderlichen *(Immutable)* Zustand zu versetzen. Das bedeutet, dass das Dateisystem auf dem Betriebssystem installiert ist, einem Schreibschutz unterliegt und keine Veränderungen, zum Beispiel anpassen von Dateien, während der Laufzeit erlaubt.

Um das Betriebssystem allerdings trotzdem nach den Anforderungen anzupassen, kann dies in der Regel über Konfigurationsdateien realisiert werden, die beim Startprozess ausgeführt werden. Anzumerken ist, dass dies jedoch von Distribution zu Distribution unterschiedlich funktioniert und gerade bei der Ersteinrichtung eine gewisse Zeit in Anspruch nehmen kann.

```
variant: fcos
version: 1.4.0
storage:
  files:
    - path: /etc/hostname
      mode: 0644
      contents:
        inline: myhostname
```

Eine exemplarische Konfigurationsdatei, die bei dem Container Operating System Fedora CoreOS den Hostname auf „myhostname“ ändert und die Berechtigungen auf die „/etc/hostname“-Datei auf 0644 setzt. [9]

Als weitere Option bieten verschiedene Distributionen die Möglichkeit, auf dem schreibgeschützten Dateisystem ein weiteres Dateisystem hinzuzufügen. Dieses zusätzliche Dateisystem kann unter anderem neue Dateien erstellen oder modifizieren und Änderungen am Betriebssystem vornehmen. Zu beachten ist jedoch, dass alle Änderungen flüchtiger Natur sind und verloren gehen, sobald die laufende Instanz neu gestartet wird.

Da sich das Dateisystem in einem unveränderlichen Zustand befindet, ist es auch nicht ohne weiteres möglich auf dem System (Sicherheits-)Updates zu installieren. Um das System dennoch auf einen aktuellen Stand zu halten, wird das veraltete Image (zum Beispiel ISO) durch ein aktuelles Images ersetzt. Dank der Konfigurationsdateien, die beim initialen Start ausgeführt werden, wird das System in den selben Zustand versetzt, wie vor dem Update.

Automatisierte Updates:
Viele Container Operating Systems, darunter Fedora CoreOS, bieten die Möglichkeit automatisierte Updates mit einer Rollback-Funktion auszuführen. Dabei prüft das Betriebssystem in einem regelmäßigen Abstand nach potenziellen Si-

cherheitspatches oder Updates, und installiert diese und fügt eine weitere Schicht hinzu. Sollte ein bereitgestellter Service nach dem Patchen beeinträchtigt sein, kann man die zuletzt installierten Sicherheitspatches und Updates entfernen, indem die zuletzt hinzugefügte Schicht entfernt wird (Rollback). [10]

Damit der Patchprozess nicht unmittelbar nachdem gestartet wird, wenn verfügbare Sicherheitspatches oder Updates gefunden werden, kann ein Zeitrahmen definiert werden, zum Beispiel täglich zwischen 22:00 – 06:00 Uhr, indem der Patchprozess ausgeführt werden darf. Da es während des Patchens zu einer gewissen Ausfallzeit kommen kann, sollten diese zu einer Uhrzeit ausgeführt werden, bei der die Auslastung möglichst gering ist. [10]

Programme die diese Funktionalität bereitstellen sind unter anderem „OSTree“ und „Zincati“. [10]

B. Absicherung des eingesetzten Images

1) Vertrauenswürdige Quellen für Images: Prinzipiell ist es jedem möglich als eine Image-Quelle zu fungieren. Aufgrund dessen muss auf die Herkunft des Image besonders geachtet werden, wie in Nicht-vertrauenswürdige Image(s) erläutert. Im Rahmen einer Analyse wurde festgestellt, dass 80 Prozent der auf Docker-Hub *(https://hub.docker.com/)* gehosteten Images teilweise schwerwiegende Sicherheitslücken aufweisen. [8] Aus diesem Grund reicht es leider nicht aus, sich für mehr Sicherheit auf offizielle Quellen wie Docker-Hub zu beschränken. Um dem entgegenzuwirken, gibt es Images, die als „offizielles Image“ gekennzeichnet sind.

Abbildung 5. In dieser Abbildung ist zu erkennen, dass es sich bei dem Alpine Image um ein „Docker Offical Image“ handelt. (https://hub.docker.com/_/alpine)

Diese Art von Image muss unterschiedliche Kriterien erfüllen, um solch eine Kennzeichnung zu erhalten. Unter anderem muss die Dockerfile, die das Image baut, vollständig dokumentiert werden, damit Nachvollzogen werden kann, warum eine Anwendung installiert und entsprechend konfiguriert ist. Darüber hinaus wird auch geprüft, ob die Best Practices (https://docs.docker.com/develop/develop-images/dockerfile_best-practices/) eingehalten werden. [11] Ein weiteres Kriterium ist, dass Image-Updates in regelmäßigen Abständen veröffentlicht werden müssen, damit unter anderem der Einsatz mit dem jeweiligen Image längerfristig geplant werden kann und unter anderem keine veralteten Images entstehen, die mit erhöhter Wahrscheinlichkeit Sicherheitslücken enthalten. [11] Aus diesen Gründen, ist es dringend zu empfehlen ein Image mit solch einer Kennzeichnung vorzuziehen.

Unter bestimmten Bedingungen bieten verschiedene Image-Anbieter CVE-Analyseberichte (siehe Kapitel II-D) an. Dies kann auch als Indikator für eine erste Einschätzung als Image-Kandidaten herangezogen werden. Um diese Funktion verwenden zu können, muss der Besitzer des Image-Registry diese jedoch aktivieren.

Die folgende Abbildung 6 zeigt, wie ein CVE-Analysebericht auf der offiziellen Docker-Hub Homepage dargestellt wird. Im oberen Abschnitt ist das aktuellste (auf Englisch: latest) Alpine-Image, welches keine bekannten CVE-Einträge enthält, zu sehen. Im unteren Abschnitt der Abbildung 6 ist ein Alpine-Image, das einen Monat älter ist als das, zum Zeitpunkt der Forschung, aktuellste Alpine-Image. Dieses Alpine-Image besitzt nach einem Monat Differenz bereits eine Sicherheitslücke vom Typ „hoch“ und zwei weitere Sicherheitslücken vom Typ „mittel“.

Anhand des Beispiels wird ersichtlich warum eine regelmäßige Aktualisierung des Images für ein sicheren Betrieb von Containern unerlässlich ist.

Es ist zu beachten, dass nicht alle Images auf beispielsweise Docker-Hub einen CVE-Analysebericht enthalten. Daher ist es ratsam, zusätzlich eine statische Image-Analyse *(siehe Kapitel IV-B2)* durchzuführen, um das Image bewerten zu können. Falls Docker-Hub für das Image ein CVE-Analysebericht bereitstellt, können diese zumindest mit der statischen Image-Analyse verglichen werden.

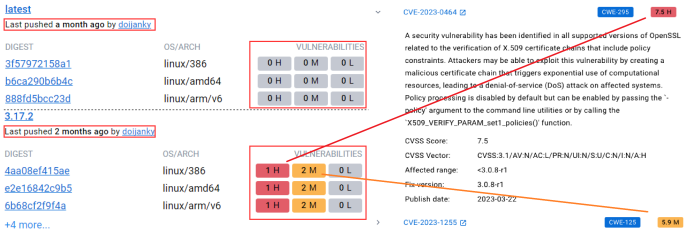


Abbildung 6. Zwei Alpine-Images die sich im direkten Vergleich befinden. Das ältere Alpine-Image weist bereits zwei Monate nach Veröffentlichung „hohe“ Sicherheitslücken auf. Durch klicken auf die Vulnerbalities gelangt man in den rechten Abschnitt der Abbildung. In diesem Bereich werden weitere Details zur Sicherheitslücke aufgeschlüsselt.

2) Statische Image-Analyse: Eine statische Image-Analyse bei Containern überprüft das Image auf potenzielle Sicherheitsrisiken, Schwachstellen, nicht genutzter oder bösartiger Software, ohne den Container selbst ausführen zu müssen.

Die statische Image-Analyse kann verschiedene Faktoren des zu analysierenden Images überprüfen, darunter das eingesetzte Basis-Image, die installierte Software oder die Konfiguration. Ein wichtiges Werkzeug, welches ebenfalls sehr beliebt ist, eine Überprüfung auf bekannte Schwachstellen und Sicherheitslücken in den installierten Anwendungen durch CVE-Scans *(siehe Kapitel II-D)*.

Der Grund eine statische Image-Analyse einzusetzen, besteht darin, frühzeitig Sicherheitsprobleme erkennen zu können,

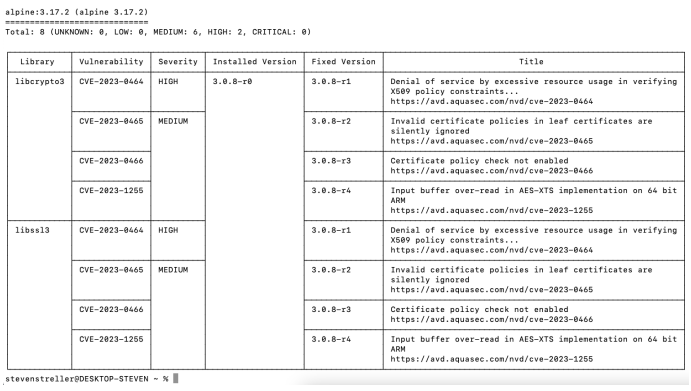
bevor das Image in einem produktiven Container eingesetzt wird. Durch dieses Wissen kann der Betreiber der Umgebung geeignete Maßnahmen ergreifen, um die Sicherheit der Umgebung zu erhöhen.

Ein Tool, welches alle Funktionalitäten bereitstellt, die in den voherigen Absätzen beschrieben wurden, heißt Trivy.

Trivy ist ein beliebtes Werkzeug für die statische Image-Analyse von Containern, welches zum aktuellen Zeitpunkt Open-Source entwickelt wird *(https://github.com/aquasecurity/trivy)*. Es unterstützt unter anderem Container-Technologien wie Docker, Kubernetes oder auch Podman.

In den folgenden Absätzen wird die konkrete Nutzung von Trivy unter Docker demonstriert. Hierfür wird das bereits vorgestellte Image `alpine:3.17.2` herangezogen.

Um ein Image mit Trivy zu scannen, muss der folgende Befehl in der CLI ausgeführt werden:
`docker run aquasec/trivy image alpine:3.17.2` Nach der Ausführung des Befehls, erhält



alpine:3.17.2 (alpine 3.17.2)
Total: 8 (UNKNOWN: 0, LOW: 0, MEDIUM: 6, HIGH: 2, CRITICAL: 0)

Library	Vulnerability	Severity	Installed Version	Fixed Version	Title
libcrypto3	CVE-2023-0464	HIGH	3.0.8-r0	3.0.8-r1	Denial of service by excessive resource usage in verifying X509 policy constraints... https://avd.aquasec.com/nvd/cve-2023-0464
	CVE-2023-0465	MEDIUM		3.0.8-r2	Invalid certificate policies in leaf certificates are silently ignored https://avd.aquasec.com/nvd/cve-2023-0465
	CVE-2023-0466			3.0.8-r3	Certificate policy check not enabled https://avd.aquasec.com/nvd/cve-2023-0466
	CVE-2023-1255			3.0.8-r4	Input buffer over-read in AES-XTS implementation on 64 bit ARM https://avd.aquasec.com/nvd/cve-2023-1255
libssl3	CVE-2023-0464	HIGH		3.0.8-r1	Denial of service by excessive resource usage in verifying X509 policy constraints... https://avd.aquasec.com/nvd/cve-2023-0464
	CVE-2023-0465	MEDIUM		3.0.8-r2	Invalid certificate policies in leaf certificates are silently ignored https://avd.aquasec.com/nvd/cve-2023-0465
	CVE-2023-0466			3.0.8-r3	Certificate policy check not enabled https://avd.aquasec.com/nvd/cve-2023-0466
	CVE-2023-1255			3.0.8-r4	Input buffer over-read in AES-XTS implementation on 64 bit ARM https://avd.aquasec.com/nvd/cve-2023-1255

stevenstreller@DESKTOP-STEVEN ~ %

Abbildung 7. Ergebnis des Befehls docker run aquasec/trivy image alpine:3.17.2

man einen Report, der die gefundenen Sicherheitslücken auflistet, die exakt die identischen sind wie in der Abbildung 6.

Es ist empfehlenswert die statische Image-Analyse kontinuierlich ausführen zu lassen, um zu überprüfen ob die eingesetzten Images, aktuell sind. Dies könnte im Rahmen einer CI/CD-Pipeline oder einem vordefinierten Prozess geschehen.

Trivy ist nicht das einzige Werkzeug, welches für die statische Image-Analyse genutzt werden kann. Weitere Tools sind unter anderem Clair, Anchore Engine oder Aqua Security.

C. Absicherung des Containers durch Konfigurationsparameter von Containern

Durch die Umsetzung einiger einfacher Maßnahmen kann der Betrieb von Containern erheblich verbessert werden. Wie im Kapitel unzureichende Konfiguration dargelegt, birgt es ein erhöhtes Risiko, Containern uneingeschränkten Zugriff auf Hardware-Ressourcen zu gewähren. Sowohl Docker als auch Kubernetes bieten jedoch Mechanismen, um die Ressourcennutzung von Containern einzuschränken.

Die Einschränkung der Ressourcennutzung kann in **Docker** ganz einfach durch das Hinzufügen entsprechender Konfigurationen zur Orchestrierungsdatei (docker-compose) des Services umgesetzt werden, was wie folgt dargestellt ist. [12]

```
services:
  my-alpine-container:
    image: alpine
    resources:
      limits:
        cpus: '0.5'
        memory: '512M'
```

Limitierung der Ressourcen in **Docker** mit folgenden Einschränkungen: „my-alpine-container"-Container kann maximal 512 Megabyte Arbeitsspeicher (Memory) und 50 Prozent des Prozessors (CPU) beanspruchen. [12]

Alternativ kann auch die Docker CLI genutzt werden: `docker run --cpus 0.5 --memory 512M alpine`

In **Kubernetes** sieht die Einschränkung der Ressourcen ähnlich aus:

```
apiVersion: v1
kind: Pod
spec:
  containers:
    - name: my-alpine-container
      image: alpine
      resources:
        limits:
          cpu: "0.5"
          memory: "512Mi"
```

Limitierung der Ressourcen in **Kubernetes** mit folgenden Einschränkungen: „my-alpine-container"-Container kann maximal 512 Megabyte Arbeitsspeicher (Memory) und 50 Prozent des Prozessors (CPU) beanspruchen. [13]

Es ist wichtig, kleinere Maßnahmen zu beachten, die dennoch von großer Bedeutung sein können. Dazu gehört beispielsweise, dass der Container, sofern nicht unbedingt erforderlich, nicht im selben Netzwerk wie das Host-System betrieben werden sollte (Host-Mode). Standardmäßig erhält jeder bereitgestellte Container in Docker sein eigenes Netzwerk und kann nicht andere Container erreichen.

Zusätzlich ist es ratsam zu evaluieren, welche Ports geöffnet werden müssen, um den Container ordnungsgemäß betreiben zu können. Es ist empfehlenswert, nur Ports zu öffnen, die wirklich für den Betrieb des Containers erforderlich sind. Die Dokumentation des Images ist in der Regel eine gute Anlaufstelle, um herauszufinden, welche Ports erforderlich sind.

Abschließend existieren Programme die genutzt werden können, um die getätigte Konfiguration der Container überprüfen und bewerten zu lassen. Eines dieser Programme wird nachfolgend vorgestellt.

„Docker Bench for Security“ ist ein Projekt, welches Open-Source entwickelt wurde, um die Sicherheitskonfiguration von der Docker-Instanz zu überprüfen. Es ist ein Skript, welches eine Abfolge von Best Practices und Sicherheitsrichtlinien anwendet, um Sicherheitslücken in der Docker-Konfiguration zu entdecken.

Das Tool führt automatisierte Tests durch, um sicherstellen zu können, dass die Umgebung sicher konfiguriert ist. Es überprüft unter anderem die Konfiguration des Host-Systems, die Netzwerkeinstellungen und andere relevante Konfigurationen. Es prüft beispielsweise, ob die neuste Version von Docker verwendet wird und ob der Benutzer angemessen berechtigt ist.

Nach der Ausführung des Skripts erhält man Informationen über Fehlkonfigurationen oder zu Empfehlungen, die umgesetzt werden sollte. Abschließend erhält man eine Schulnote (1-6), die die Sicherheit der Konfiguration widerspiegelt.

„Docker Bench for Security” hilft dabei die Umgebung auf Sicherheitsprobleme zu überprüfen und daraus resultierende Maßnahmen ergreifen zu können, um eine angemessene Sicherheitskonfiguration gewährleisten zu können.

Es ist auf jeden Fall empfehlenswert das Tool in regelmäßigen Abständen auszuführen, um das Risiko einer fehlerhaften Konfiguration zu vermeiden.

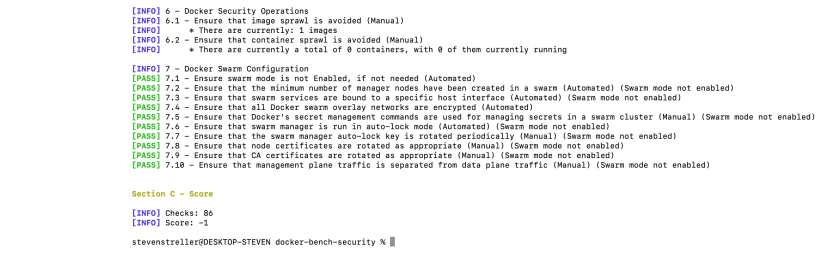


Abbildung 8. Ein Teilausschnitt, was „Docker Bench for Security“ für hilfreichen Inhalt ausgibt. Abschließend wird eine Note der Konfiguration vergeben

V. Zusammenfassung und Ausblick

A. Zusammenfassung

Der sichere Betrieb von Containern ist von großer Bedeutung und erfordert eine umfassende Herangehensweise. In dieser Arbeit wurden wichtige Aspekte zur Gewährleistung der Containersicherheit betrachtet.

Zu Beginn wurde auf potenzielle Risiken hingewiesen, die mit dem Betrieb von Containern verbunden sein können. Insbesondere wurde betont, wie wichtig es ist, jedes Image vor der Verwendung in einer Produktivumgebung sorgfältig zu evaluieren. Dies beinhaltet die Auswahl einer vertrauenswürdigen Quelle sowie Autors des Images, die Überprüfung von CVE-Analyseberichten und die Durchführung einer statischen Image-Analyse.

Des Weiteren wurde die Bedeutung eines effektiven Patchmanagements vorgestellt. Sowohl das Host-Betriebssystem als auch die laufenden Container-Instanzen können Sicherheitslücken aufweisen, wenn ein unzureichendes Patchmanagement vorhanden ist. Vorgestellte Maßnahmen, sind die Implementierung von automatisierten Updates mit Rollback-Funktion.

Abschließend wurde auf die Auswirkungen einer unzureichenden Konfiguration hingewiesen und Möglichkeiten aufgezeigt, wie diese vermieden werden können.

Die in der Arbeit vorgestellten Aspekte bieten einen Leitfaden für den sicheren Betrieb von Containern sowie die Möglichkeit durch Anwendung der Maßnahmen einen soliden Sicherheitsansatz, und daraus folgend eine sichere und geschützte Container-Umgebung gewährleisten zu können.

B. Limitationen und zukünftige Forschungsrichtungen

Diese Arbeit entstand im Rahmen einer Seminararbeit in dem Master-Studiengang „Angewandte Informatik“. Innerhalb des Seminars wurden verschiedene Sicherheitsrelevante Aspekte beleuchtet, darunter der sichere Betrieb von Linux-Servern und die sichere Entwicklung von Software in Python.

Das Seminar hatte einen festen Zeitrahmen, so dass abgewogen werden musste, welche Aspekte in der Seminararbeit vorgestellt und diskutiert werden.

Eine weitere Maßnahme die definitiv näher betrachtet werden sollte, ist das Protokollieren von Ereignissen und einer anschließenden Analyse um potenzielle Angriffe erkennen zu können.

Literatur

[1] Unknown. *CNCF SURVEY 2020*. CNCF, 2020.

[2] Aditya Bhardwaj and C. Rama Krishna. Virtualization in cloud computing: Moving from hypervisor to containerization—a survey. *Arabian Journal for Science and Engineering*, 46(9):8585–8601, Sep 2021.

[3] Docker HTTPD (Apache-Webserver) Image. https://hub.docker.com/_/httpd. Eingesehen am: 10.04.2023.

[4] Lennart Espe, Anshul Jindal, Vladimir Podolskiy, and Michael Gerndt. Performance evaluation of container runtimes. In *CLOSER*, pages 273–281, 2020.

[5] Ubuntu Manpage: namespaces. https://manpages.ubuntu.com/manpages/xenial/en/man7/namespaces.7.html. Eingesehen am: 29.04.2023.

[6] Soonhong Kwon and Jong-Hyouk Lee. Divds: Docker image vulnerability diagnostic system. *IEEE access*, 8:42666–42673, 2020.

[7] What is a CVE? https://www.redhat.com/en/topics/security/what-is-cve. Eingesehen am: 28.04.2023.

[8] Oliver Liebel. *Skalierbare Container-Infrastrukturen : das Handbuch für Administratoren (3. Auflage)*. Rheinwerk Verlag, Bonn, Deutschland, 2021.

[9] Fedora Docs. https://docs.fedoraproject.org/en-US/fedora-coreos/hostname/. Eingesehen am: 28.04.2023.

[10] Fedora CoreOS Docs: Auto-Updates and Manual Rollbacks. https://docs.fedoraproject.org/en-US/fedora-coreos/auto-updates/. Eingesehen am: 30.04.2023.

[11] Docker Official Images. https://docs.docker.com/docker-hub/official_images/. Eingesehen am: 04.05.2023.

[12] Runtime options with Memory, CPUs, and GPUs. https://docs.docker.com/config/containers/resource_constraints/. Eingesehen am: 04.05.2023.

[13] Resource Management for Pods and Containers. https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/. Eingesehen am: 04.05.2023.