INSTITUTE FOR NETWORKED SOLUTIONS

OST Ostschweizer Fachhochschule

Container Security

Studienarbeit

Studiengang Informatik

OST – Ostschweizer Fachhochschule

Campus Rapperswil-Jona

Herbstsemester 2020

Autor:	Fabio Caspani
Betreuer:	Prof. Dr. Nathalie Weiler
Projektpartner:	INS Institute for Networked Solutions, Rapperswil

1 Aufgabenstellung

1.1 Einführung

Für Container Security gibt es mittlerweile einige vielversprechende open source auditing Tools (Docker Bench for Security, Clair, Cilium, Anchor, etc.). Dies Tools sind so ausgelegt, dass sie dem Entwickler helfen sollen potentielle Sicherheitsprobleme in seinem Container zu finden. Der Vorteil wäre, dass so Sicherheitslücken früh (und damit günstig) beseitigt werden können, anstelle dass sie kurz vor Produktionseinführung erst entdeckt werden.

1.2 Aufgabe

In dieser Arbeit sollen die Eignung dieser Tools zur automatisierten Behebung von Security Issues in Containern evaluiert und illustriert werden.
Als Minimalziel soll anhand der CIS Benchmark für Docker anhand einer Beispielapplikation in einer Docker gezeigt werde wie Vulnerabilities der Kategorien Host Configuration und Docker Daemon Configuration behoben werden können.

1.3 Hinweise

Als Start wird empfohlen sich eine Übersicht über die CIS Docker Benchmark zu verschaffen:
https://www.cisecurity.org/benchmark/docker/
Desweiteren sind unter den Tools folgende empfohlen zur Analyse:

- https://github.com/docker/docker-bench-security
- https://github.com/aquasecurity/docker-bench
- https://github.com/quay/clair

Sowie ein Tutorial für ubuntu unter https://www.digitalocean.com/community/tutorials/how-to-audit-docker-host-security-with-docker-bench-for-security-on-ubuntu-16-04

1.4 Erwartete Resultate

- Software welche die Anwendung der CIS Benchmark für Docker zeigt.
- Dokumentation der Arbeit (inklusive Vorgehen und kritische Bewertung der getroffenen Entscheide)

Prof. Dr. Nathalie Weiler ■ nathalie.weiler@ost.ch ■ T +41 55 222 41 19
OST – Ostschweizer Fachhochschule ■ INS ■ Oberseestrasse 10 ■ CH-8640 Rapperswil
T +41 58 257 41 11 ■ F +41 58 257 44 00 ■ ins@ost.ch ■ https://www.ins.hsr.ch

2 Organisatorisches

2.1 Bewertung

Die Bewertung der Arbeit erfolgt nach dem für Studienarbeiten vorgegebenen Bewertungsschema (Gewichtung in Klammern):

1. (1/5) Organisation und Durchführung: Projektplanung und Nachführung der Arbeit gemäss Projektplan, Zusammenarbeit mit dem allfälligen Auftraggeber und der Betreuerin
2. (1/5) Bericht: Inhalt des Projektschlussberichtes, Gliederung, Darstellung und Sprache der Dokumentation
3. (3/5) Resultat der Arbeit:
    a. Problemanalyse: Vorstudie, Literaturstudium, Anforderungsspezifikation, Anforderungsanalyse
    b. Lösungsentwurf: Lösungsvarianten und deren Beurteilung, Variantenentscheid, Konzept und Entwurf
    c. Realisierung und Test

2.2 Verfassen der Arbeit

Prof. Dr. Zimmermann hat eine Zusammenstellung erstellt mit Hinweisen zum Schreiben einer wissenschaftlichen Arbeit: https://ifs.hsr.ch/index.php?id=13194&L=4metadata%2Foai_dc_1,doc

Weitere Angaben zur Studienarbeit im allgemeinen befinden sich auf dem Skripte-Server unter \\hsr.ch\root\alg\skripte\Informatik\Fachbereich\Studienarbeit_Informatik\

2.3 Termine

Die Studienarbeit beginnt am 14. September 2020.
Abgabetermin ist der 18. Dezember 2020 bis 17 Uhr.

2.4 Betreuung

Die Studienarbeit wird durch Prof. Dr. Nathalie Weiler betreut.
Eine regelmässige, wöchentliche Besprechung wird zwischen Studierenden und Betreuerin am Anfang der Arbeit festgelegt: Mittwoch, 10-11 Uhr

Ausgabe: Rapperswil, 16. September 2020

Prof. Dr. Nathalie Weiler
Institutspartnerin INS

Prof. Dr. Nathalie Weiler ■ nathalie.weiler@ost.ch ■ T +41 55 222 41 19
OST – Ostschweizer Fachhochschule ■ INS ■ Oberseestrasse 10 ■ CH-8640 Rapperswil
T +41 58 257 41 11 ■ F +41 58 257 44 00 ■ ins@ost.ch ■ https://www.ins.hsr.ch

Abstract

In dieser Arbeit wurde eine Software entwickelt, um Sicherheitsrisiken von Applikationscontainern zu veranschaulichen, da in den letzten Jahren die Container stark an Bedeutung gewonnen haben. Einige der häufigsten Sicherheitsrisiken von Docker Containern werden in einer Demonstrationssoftware aufgezeigt. Der Umfang der Demonstrationssoftware wurde in einer Vorstudie beleuchtet und eingegrenzt. Im Zusammenhang mit der Demonstrationsumgebung wurde das automatisierte Container Auditing Programm «Docker Bench for Security» untersucht.

Die entwickelte Demonstrationssoftware umfasst sieben allgemeine Sicherheitsrisiken, von unsicheren Netzwerken bis zu Ausbrüchen aus dem Container, welchen Docker Container ausgesetzt sein können. Der Anwender kann in der Software jeweils die Vulnerabilität ausnutzen und die Sicherheitslücke anschliessend beheben. Das Container Auditing Programm «Docker Bench for Security» erwies sich in dieser Form als ungeeignet für die Sicherheitsprüfung von Containern. Da alle Container eines Hosts gleichzeitig statisch analysiert werden, entfällt die Möglichkeit, auf die Sicherheitsanforderungen der einzelnen Container einzugehen. Die Audit Software macht zudem auch keine Gewichtung der einzelnen Tests. Dies kann zu einer massiven Verfälschung des Gesamtergebnisses führen. Ebenfalls wird die Auswertung unübersichtlich, da eine visuelle Unterscheidung fehlt. Sowohl betreffend Prioritäten wie auch Kategorien der Testfälle. Es wird deshalb empfohlen, eigene Sicherheitsstandards zu entwerfen und diese den Bedürfnissen des Projekts anzupassen. In der Zukunft ist es möglich, die Demonstrations-software weiter auszubauen oder eine Gewichtung in die Audit Software zu implementieren.

Danksagung

An dieser Stelle möchte ich mich bei den Personen bedanken, welche mich während der Semesterarbeit unterstützt haben.

Mein besonderer Dank geht an meine Betreuerin Frau Prof. Dr. Nathalie Weiler. Sie hat mir die Möglichkeit gegeben, die Arbeit allein durchzuführen, nachdem mein Kommilitone nicht an der Arbeit teilnehmen konnte. Während den wöchentlichen Besprechungen erhielt ich stets konstruktive und ehrliche Rückmeldungen. Wenn ich irgendwo nicht weiterkam, hatte sie jeweils den richtigen Wink, mit dem ich weiterfahren konnte.

Des Weiteren möchte ich mich bei meiner Familie bedanken, welche mich immer motiviert und Unterstützt hat.

Inhalt

1 Management Summary

1.1 Ausgangslage

Gemäss dem Sysdig 2019 Container Usage Report verdoppelt sich die Dichte der Applikations-Container jedes Jahr. Zwei Drittel davon haben signifikante Sicherheitslücken. Aus diesem Grund schafft diese Arbeit einen Überblick über einige Sicherheitsrisiken von Docker Containern. Docker ist mit einem Marktanteil von 79 Prozent der klare Marktführer bei Applikationscontainern (Sysdig, 2019).

1.2 Vorgehen

Zuerst wurden verschiedene Vulnerabilitäten von Docker Containern gesucht. Im Anschluss wurden ein Threat-Model und ein Attack-Tree gezeichnet, um Abhängigkeiten zwischen Vulnerabilitäten zu erkennen. Mit einer Demonstrationssoftware wurden die Vulnerabilitäten simuliert. Anhand der Erkenntnisse aus dem Threat-Model und dem Attack-Tree wurden verschiedene Use Cases erfasst, welche in der Demonstrationssoftware umgesetzt werden konnten. An der Demonstrationssoftware wurde das Tool «Docker Bench for Security» getestet, welches eine automatisierte Sicherheitsprüfung an den Containern durchführt.

1.3 Ergebnisse

Es resultierte eine Applikation, mit welcher Docker Container Vulnerabilitäten simuliert werden können. Die Applikation ermöglicht dem Anwender eine beschriebene Vulnerabilität auszunutzen und anschliessend die Sicherheitslücke zu schliessen. Bei der Implementierung der Use Cases hat sich gezeigt, dass ungesicherte Container einfach zu brechen sind, diese aber mit wenig Aufwand geschützt werden können. «Docker Bench for Security» eignet sich jedoch nicht für die Inspektion von Containern. Es behandelt das Gesamtsystem, statt die einzelnen Container auf ihre Risiken und Anforderungen zu überprüfen. Zudem kann der Gesamtscore verzerrt werden, da jedes Risiko gleich gewichtet und gewertet wird. Deshalb empfiehlt es sich, einen eigenen Sicherheitsstandard für Container zu erarbeiten und diesen den Bedürfnissen der jeweiligen Projekte anzupassen.

1.4 Ausblick

Um einen besseren Überblick über die Sicherheitsrisiken von Containern zu erhalten, könnte die Applikation mit weiteren Vulnerabilitäten ausgestattet werden oder die bestehenden könnten ausgebaut werden. Wenn die Container automatisiert getestet werden sollen, so sollte ein eigenes Testskript geschrieben werden, welches den eigenen Anforderungen entspricht.

2 Technischer Bericht

2.1 Einleitung

Nicht bloss im Gütertransport sondern auch in der Informatik sind Container nicht mehr wegzudenken. Gemäss dem Sysdig 2019 Container Usage Report verdoppelt sich die Dichte der Applikations-Container jedes Jahr. Dabei ist Docker der klare Marktführer mit einem Marktanteil von zirka 79 Prozent (Sysdig, 2019). Speziell im Zusammenhang mit Cloud Services und verteilten Software Systemen gewinnen Virtualisierungen mit Containern zunehmend an Bedeutung.

Bevor es die Container Technologie gab, wurden die Virtualisierungen mit virtuellen Maschinen realisiert. Virtualisierungen werden gemacht, um Applikationen kosteneffizient zu separieren. Anstatt für jede Applikation eine eigene Hardware zu beschaffen, kauft man einen grossen, leistungsfähigen Server und unterteilt diesen in verschiedene kleinere, virtuelle Server. Hier kommt die «Economy of Scale» zum Tragen. Eine virtuelle Maschine stellt, logisch gesehen, einen kompletten Rechner dar. Das bedeutet, die Systemressourcen (Rechenleistung, Arbeitsspeicher und Festplattenspeicher) werden beim Starten für die virtuelle Maschine reserviert und sind danach unabhängig vom Host. Die vollständige Isolation einer virtuellen Maschine ist ein Vorteil, jedoch muss für jede Maschine eine Reserve angelegt werden. Bei Docker laufen alle Container auf dem gleichen Betriebssystem und teilen sich somit die Systemressourcen. Dies führt zu weniger Overhead, da die Reserve nur für das Gesamtsystem beachtet werden muss. Zudem sind Container meist kleiner als virtuelle Maschinen, da nicht jeder Container ein Betriebssystem benötigt. Somit können Container auch schneller gestartet und gestoppt werden, was wiederum eine dynamische Skalierung ermöglicht.

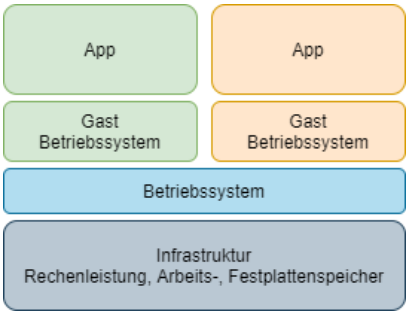

Abbildung 2: Architektur einer VM-Umgebung

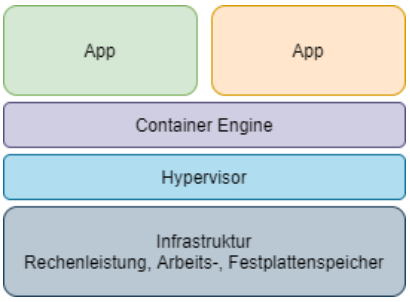

Abbildung 1: Architektur einer Container-Umgebung

Alle diese Vorteile von Containern bringen auch Nachteile gegenüber virtuellen Maschinen. Da die Container ihre Vorteile vor allem in der Flexibilität haben, leidet die Sicherheit. Wenn Container erstellt und laufen gelassen werden, so gilt es einige Punkte zu beachten. Das Thema der Containersicherheit ist aber noch nicht weit verbreitet, da Container noch eine junge Technologie ist und der Fokus momentan auf der Umsetzung liegt. Dies zeigt auch der Sysdig 2019 Container Usage Report, welcher besagt, dass noch immer zwei Drittel aller Container grosse Sicherheitsrisiken aufweisen, welche eigentlich einfach zu beheben wären (Sysdig, 2019). Um dies zu überprüfen gibt es einige Tools, welche auf einem Host die Container überprüft und bewertet.

In dieser Arbeit werden Vulnerabilitäten von Docker Containern und der Docker Engine gesucht und untersucht, wie diese vermieden werden können. Dazu werden in einem ersten Schritt nach verschiedenen Vulnerabilitäten gesucht. Um die Sicherheitsrisiken demonstrieren zu können, soll eine Software erstellt werden. Darin sollen die einzelnen Vulnerabilitäten demonstriert werden und auch, wie diese zu beheben sind. Zum Schluss soll analysiert werden, wie sich automatisierte Tools eigenen, um die Container auf Sicherheitsrisiken zu überprüfen.

Das Dokument ist so aufgebaut, dass zuerst eine Einführung in Docker gegeben wird. Danach wird das Vorgehen der Vorstudie erläutert. Dies beinhaltet das Vorgehen bei der Auswahl der zu behandelnden Vulnerabilitäten, sowie die Planung der Demonstrationssoftware. Anschliessend werden die ausgewählten Bedrohungen dokumentiert und zum Schluss folgt das Fazit. In der Software werden die ausgewählten Container-Vulnerabilitäten vorgestellt. Dabei kann der Anwender die Angriffs- und Abwehrszenarien durchspielen.

2.2 Einführung in Docker

2.2.1 Allgemein

Docker ist eine freie Container-Orchestrierung-Software. Sie verwaltet die einzelnen Docker Komponenten und bildet die Schnittstelle zu den Systemressourcen.

2.2.2 Image

Ein Image ist ein Abbild eines bestimmten Systems zu einem bestimmten Zeitpunkt. Im Zusammenhang mit Docker ist meist ein Image eines Containers gemeint.

2.2.3 Container

Container sind die zentralen Instanzen von Docker. Alle Services von Docker dienen dazu, Container zu verwalten. Container werden immer aus Images gebildet. Einzelne Container können beliebig gestartet oder gestoppt werden. Die Container sind von Docker aus gesehen komplett autonom.

2.2.4 Analogie zu Güterverkehr

Die Analogie zu Container aus dem Schiffsverkehr ist hier sehr treffend und kann für das Verständnis hilfreich sein. Die Idee hinter Software-Containern ist dieselbe, wie bei Güter-Containern. Es geht um eine standardisierte Verpackung, welche auf verschiedene Träger passt. (ISO-Container passt auf verschiedene Transportmittel; Docker Container läuft auf jedem Docker Host, unabhängig des Hostsystems). In diesem Beispiel wäre Docker ein Containerschiff, das Image eine auszuliefernde Fracht und ein Software-Container ein ISO-Container.

Wenn man eine materielle Fracht ausliefern muss, so packt man diese für den Transport in einen ISO-Container. So muss man sich nicht um die Einzelheiten der Transportmittel kümmern, solange diese ISO-Container unterstützen. Gleich verhält es sich bei Software. Dort liefert man die Software auch in einem Container aus, damit man sich nicht um die Spezifikationen des Host Systems kümmern muss.

2.2.5 Netzwerke

Docker verwaltet auch eigene Netzwerke für die Container. Jeder Container läuft in einem Docker Netzwerk, welches manuell oder automatisch zugewiesen wurde. Innerhalb eines Netzwerks können Container miteinander kommunizieren. Zwischen zwei verschiedenen Netzwerken ist keine Kommunikation möglich.

Wenn ein Port bei einem Container geöffnet werden soll, so muss dies explizit angegeben werden. Hier kann definiert werden, ob der Port nur innerhalb des Netzwerks erreichbar ist, oder ob er auch vom Host erreicht werden kann.

2.2.6 Volumes

Um Daten in Docker persistent zu speichern sind sogenannte Volumes nötig. Grundsätzlich ist jeder Container flüchtig und somit auch dessen Daten. Volumes können unabhängig von Containern erstellt werden und werden dann einem oder mehreren Containern zugewiesen.

2.2.7 Dockerfile

In einem Dockerfile können verschiedenste Parameter zum Erstellen eines Images oder Containers spezifiziert werden. Wenn neben Dockerfiles auch Compose verwendet wird, so dient das Dockerfile lediglich dazu, das Image zu erstellen.

2.2.8 Compose

Docker Compose wird hauptsächlich verwendet, um die Zusammenhänge zwischen verschiedenen Containern zu definieren. Mit Compose können mehrere Container gleichzeitig gestartet werden, es können Netzwerke definiert und zugewiesen, sowie Abhängigkeiten definiert werden.

2.3 Vorstudie

2.3.1 Vorgehen bei Vulnerabilitäten suchen

Als Grundlage bei der Suche nach Vulnerabilitäten in Docker diente die CIS Benchmark. Da diese aber 115 Sicherheitsempfehlungen auflistet, musste die Auswahl noch eingeschränkt werden. Dazu wurde die «OWASP Docker Top 10»[1] verwendet. OWASP (Abkürzung für Open Source Web Application Security) ist eine gemeinnützige Stiftung, die sich für die Verbesserung der Sicherheit von Software einsetzt (OWASP Foundation, 2020).

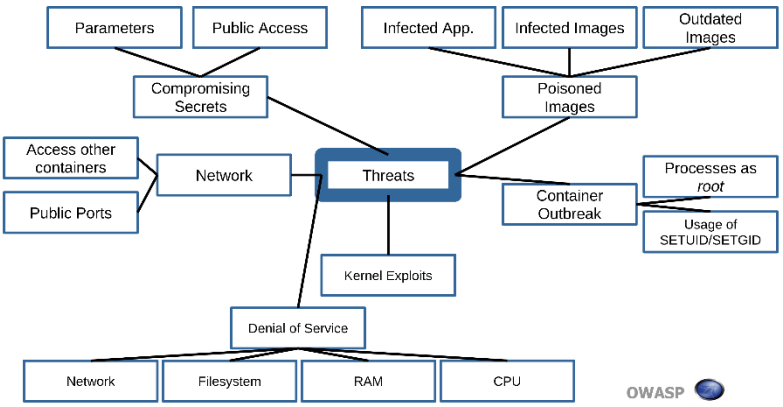

Abbildung 3: Docker Security Threats Quelle: (OWASP, 2019)

OWASP hat ein Übersichtsdiagramm der verschiedenen Bedrohungen erstellt und diese dabei kategorisiert. Daraus wurde eine Liste mit den Bedrohungen erstellt. Zu jeder Bedrohung wurde aufgelistet, welche Voraussetzungen dafür gegeben sein müssen, wie ein potenzieller Angriff ablaufen könnte und wie dieser abgewehrt werden kann. Zusätzlich wurde die Verbindung zur CIS Benchmark gesucht.

[1] OWASP Docker Top 10: https://github.com/OWASP/Docker-Security [02.12.2020]

Threat	Precondition	Angriff	Abwehr	Docker Benchmark
Container Outbreak				
Insecure User Mapping	Container läuft in unter `root` oder `--privileged`	Wenn Applikation eine Schwachstelle hat, kann ein Angreifer auch aus der Applikation ausbrechen	Nie `--privileged` flag benutzen `RUN useradd <username>` Ports mappen	5.4 4.1
Authorization for Docker Client	-	any user with permission to access the Docker daemon can run any Docker client command	Authorization Plugin	2.11
Network				
Public Access	Mehrere Interfaces auf Host	Container, welcher für eine sichere Umgebung ausgelegt ist, kann von einem "public" Interface erreicht werden	Container Port an Host/Schnittstelle und Port binden	5.13
Unrestricted Traffic between Containers	Mehrere Container auf Host	Container können per default uneingeschränkt miteinander kommunizieren	Restrict traffic between Containers `--icc`	2.1
TLS for Docker Daemon	Docker Deamon is exposed via Network	Jeder, der den Port kennt hat Zugang zum Daemon	Nur Authentisierte Verbindungen Zulassen mit TLS	2.6
Compromising Secrets				
Secrets in Dockerfile	Secret stored in Dockerfile	Jeder User kann das Dockerfile lesen	Use a Secrets Management Tool if nessecary	4.10
Poisoned Images				
Outdated Images	Image not up to date	Bekannte Schwachstelle im alten Image ausnutzen	Explizite Versionisierung verwenden	5.27
Infected Images	Image from untrusted Source	Fake Image mit eingebauter Schwachstelle	Use only Trusted Sources	
Vulnerable Application				
CSRF	1. Das Opfer muss authentisiert sein (Session Cookie) 2. Es muss ein XSRF-Link vorhanden	Opfer ruft Link auf Angreifer-Webseite auf, welcher eine Aktion auf vulnerabler Webseite ausführt	Tokens für Transaktionen SameSite Cookie Attribute	-
XSS	Keine Inputvalidierung	Session Hijacking Stored XSS	Secure Programming	-
Denial of Service				
CPU	Es sind mehrere Container auf einem Host	Eine Applikation belastet alle CPUs zu 100%	Maximale CPU Belegung beschränken `--cpus=<value>` Flag	5.11
RAM	Es sind mehrere Container auf einem Host	Eine Applikation belegt den gesamten Memory	Memory pro Container beschränken `--memory` flag	5.10
Storage	Es sind mehrere Container auf einem Host	Eine Applikation füllt den gesamten Storage	-	
Network	Es sind mehrere Container auf einem Host	Eine Applikation legt das Netzwerk lahm	Load Balancing	

Abbildung 4: Übersicht über die Vulnerabilitäten

Um die geplante Demonstrationssoftware umsetzten zu können, brauchte es noch eine verwundbare Docker Applikation. Dazu wurde nach einem Open Source und frei verfügbaren Webshop gesucht. ExpressCart[2] erwies sich dabei als die ideale Software. Es ist ein Webshop, welcher frei auf Github verfügbar ist. Die Applikation ist in Node.js geschrieben und nicht besonders gross oder komplex. Dies vereinfacht kleinere Änderungen an der Applikation und erlaubt, dass die Applikation schnell gestartet werden kann. Eine zu komplexe Demonstrationssoftware würde am Ziel dieser Arbeit vorbeischiessen.

[2] Express Cart: https://github.com/mrvautin/expressCart [29.11.2020]

2.3.2 Threat Model beschreiben

Aufgrund der aufgelisteten Sicherheitsrisiken wurde ein Architektur-Threat-Modell gezeichnet. Anhand dieses Modells sollen die Bedrohungen, welche einen Zusammenhang mit der Architektur haben, aufgezeigt werden. Zudem kann so die spätere Architektur der Demonstrationsumgebung bestimmt werden.

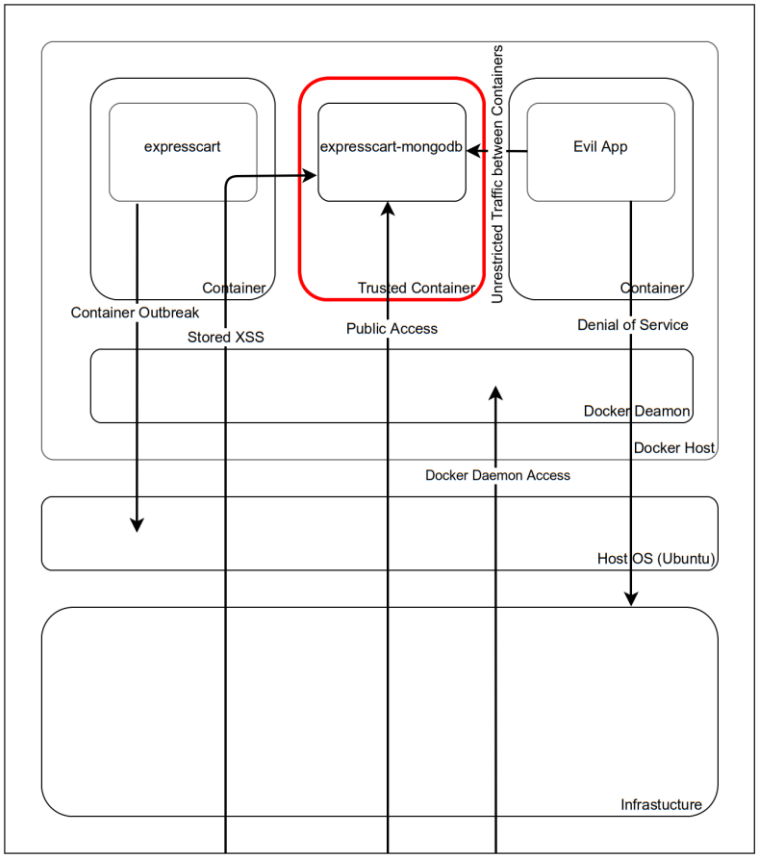

Abbildung 5: Threat Model

Das Modell stellt einen Server mit installierter Docker Engine dar. In Docker läuft die Express Cart Software. Dies sind die beiden linken Container «expresscart» und «expresscart-mongodb». Um die einen Bedrohungen simulieren zu können, benötigt es noch einen zusätzlichen Container mit einer Applikation, welche eine verwundbare oder böse Applikation auf dem gleichen Host darstellt.

2.3.3 Attack Tree beschreiben

Um die Abhängigkeiten zwischen den verschiedenen Bedrohungen besser zu verstehen, wurde ein Attack Tree gezeichnet. Dieser wurde in zwei grundlegende Kategorien aufgeteilt. Auf einer Seite stehen die Vulnerabilitäten, welche auf eine unsichere Umgebung zurückgeführt werden können. Auf der anderen Seite stehen die Vulnerabilitäten, welche mit der Programmierung der Applikation zusammenhängen. Die beiden Attacken (XSRF und XSS) unter «Vulnerable App» sind lediglich zwei Beispiele unter unzähligen. Diese fallen jedoch unter das Thema der Applikationssicherheit und werden deshalb in dieser Arbeit weniger stark gewichtet. Eine kurze Sicherheitsprüfung von Express Cart hat ergeben, dass die Applikation anfällig gegen XSRF (Cross Site Request Forgery) und XSS (Cross Site Scripting) ist.

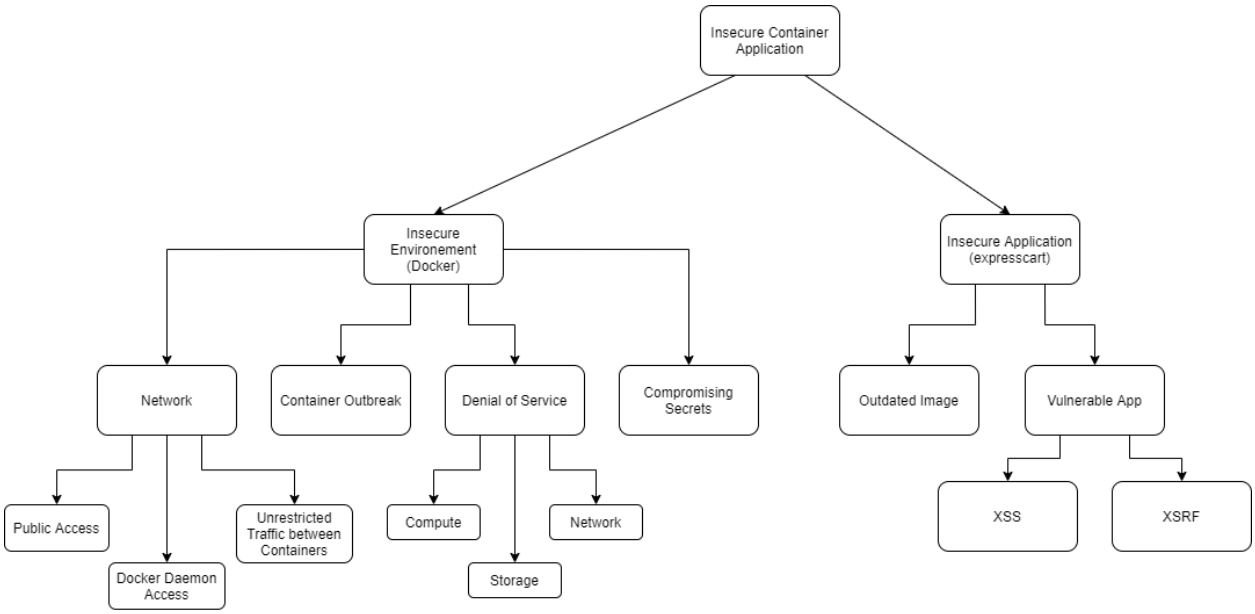

Abbildung 6: Attack Tree

2.3.4 Architekturen

System

In der Demonstration-Umgebung soll ein Server, mit mehreren laufenden Containern, simuliert werden. Um den Aufwand dafür möglichst klein zu halten, wird ein Ubuntu als virtuelle Maschine aufgesetzt. Damit die virtuelle Maschine eine Schnittstelle im LAN der Host Maschine erhält, muss eine Netzwerkbrücke errichtet werden. Zudem wird Docker und Docker Compose auf das Ubuntu installiert.

Applikationen

Auf der virtuellen Maschine laufen zwei Applikationen in Docker. Eine Applikation dient als «Opfer»-Applikation, die zweite nimmt die Rolle des «Angreifers» ein. Beide Applikationen werden nur soweit erklärt, wie es für die Containersicherheit nötig ist.

Express Cart

Express Cart ist eine Open Source Web-Shop Applikation, welche mit einer MIT[3] Lizenz auf Github zur Verfügung steht. Sie besteht aus einem Container für die Logik und einem Container mit einer Mongo Datenbank. Die beiden Container laufen im selben Docker Netzwerk. Die Applikation ist in Node.js geschrieben und die Verbindung zur Datenbank erfolgt nicht authentifiziert. In der Demonstrationsumgebung dient sie als «Opfer», um Verwundbarkeiten von Containern zu demonstrieren. Im Anschluss werden die Sicherheitslücken, in Bezug auf Container Sicherheit, geschlossen.

Angreifer App

Die Angreifer App ist eine einfache Node.js Applikation, welche im Rahmen dieser Arbeit entwickelt wurde. Sie läuft in einem eigenen Container, welcher sich ebenfalls in einem eigenen Docker Netzwerk befindet. Wie es der Name bereits verrät, dient sie als Angreifer in der Demonstrationsumgebung. Von der Applikation aus, werden verschiedene Attacken gegen Express Cart oder die virtuelle Maschine gefahren. Im Anschluss werden Massnahmen aufgezeigt, wie man sich gegen diese Attacken schützen könnte.

Zudem dient die Angreifer App auch als Anleitung für die verschiedenen Aufgaben. Dafür wurden die Readme.md Dateien der einzelnen Aufgaben in die Applikation kopiert. Damit die Markdown Datei gut gestylt ist, wurde das vorgefertigte CSS-Stylesheet «github-markdown-css»[4] verwendet.

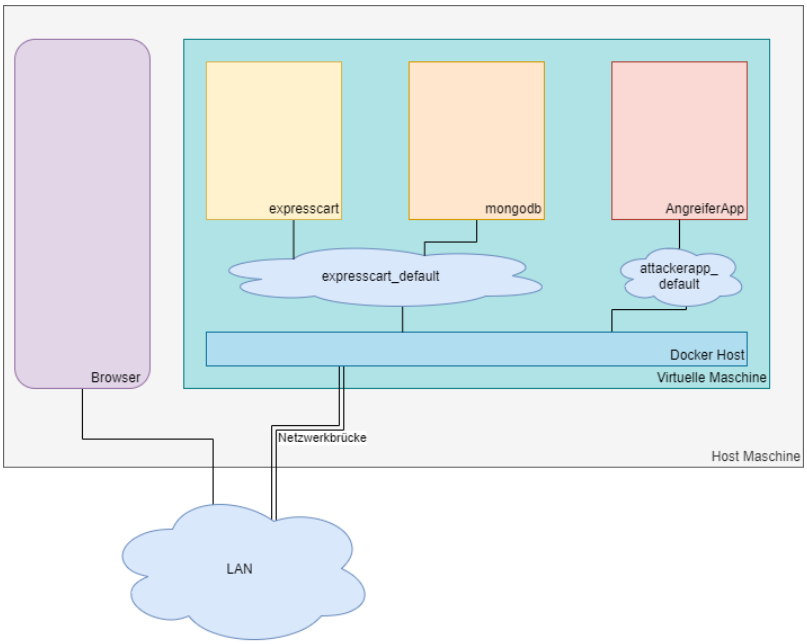

Abbildung 7: Übersichtsschema Demonstartions-Umgebung

[3] MIT Lizenz: erlaubt Verwendung, Veränderung, Veröffentlichung und kommerzielle Nutzung der Software
[4] Github-Markdown-CSS: https://github.com/sindresorhus/github-markdown-css [25.11.2020]

2.3.5 Planung der Demo-Software

Das Architektur Threat Modell hat ergeben, dass neben der verwundbaren Express Cart Applikation noch eine zweite Applikation nötig ist, welche Attacken auf die Express Cart Applikation und das Host System durchführt. Sie wird «Angreifer App» genannt. Die Angreifer App soll ebenfalls in Node.js programmiert werden.

Um die Demonstrationssoftware zu planen, wurden konkrete Use Cases definiert. Anhand dieser Use Cases wird die Applikation programmiert. Die Use Cases dienen auch dazu, den Fortschritt während der Arbeit zu messen und den Aufwand zu planen. In der Umsetzung hat sich gezeigt, dass ein Fortschritt von zirka zwei Use Cases pro Woche sinnvoll ist.

Auf Unit- und Integrationstests wird bewusst verzichtet, da die Zeit knapp ist und die Applikation keine komplexe Logik hat. Es werden auch keine Edge-Cases abgefangen, da die Applikation nur zu Demonstrationszwecken für technisch versierte Anwender gedacht ist. Falls die Applikation abstürzt, kann der Anwender diese neu starten. Zum Schluss wird ein Systemtest gemacht, wobei für jeden Use Case das Main Success Scenario durchgespielt und dokumentiert wird.

Das Ziel war es die Use Cases 1 bis 7 während der Arbeit umzusetzen. Die Use Cases 8 und 9 wurden zur Reserve geplant, falls die Implementation schneller voranschreitet als geplant. Schlussendlich lief die Integration nach plan ab und die Use Cases 1 bis 7 konnten implementiert werden, für die Use Cases 8 und 9 blieb keine Zeit mehr.

Use Case Nummer:	0
Use Case Name:	ExpressCart aufsetzen & Angreifer App “Hello World”
Primärer Aktor:	
Kurzbeschreibung:	
Main Success Scenario:	1. ExpressCart läuft in Docker 2. Beispieldaten sind in ExpressCart geladen 3. Ein «Hello World» der Angreifer App läuft in Docker
Mögliche Erweiterungen:	

Use Case Nummer:	1
Use Case Name:	Unrestricted Traffic between Containers
Primärer Aktor:	Angreifer App & Benutzer
Kurzbeschreibung:	Die Angreifer App kann standardmässig auf alle anderen Container zugreifen. Somit auch auf die DB von ExpressCart. Dies soll demonstriert werden, indem man durch einen Button in der Angreifer App auf die Daten zugreifen kann. Anschliessend soll diese Sicherheitslücke geschlossen werden.
Main Success Scenario:	1. Die ExpressCartDB ist nicht gegen inter Container Traffic geschützt. 2. Der Benutzer löst bei der Angreifer App eine Datenbankabfrage aus. 3. Die Angreifer App lädt Daten aus der ExpressCartDB und zeigt diese dem Benutzer an. 4. Der Benutzer schliesst die Sicherheitslücke. 5. Die Angreifer App erreicht die Datenbank nicht mehr. 6. Die Angreifer App zeigt dem Benutzer die Fehlermeldung.
Mögliche Erweiterungen:	Der Datenbankzugriff und die Abwehrmassnahmen sind auf der Angreifer App dokumentiert und visualisiert.

Use Case Nummer:	2
Use Case Name:	Public Access
Primärer Aktor:	Externer Angreifer
Kurzbeschreibung:	Innerhalb der Docker-Umgebung werden standardmässig alle Container gleichbehandelt. Das bedeutet, dass Container, welche in einer «gesicherten» Umgebung betrieben werden, auch von «public» Interfaces erreicht werden können.
Main Success Scenario:	1. Der Container der DB ist an die Wildcard-Adresse 0.0.0.0 gebunden. 2. Der Benutzer lädt Daten aus der Datenbank. 3. Der Benutzer schützt den Datenbank-Container gegen Zugriffe von ausserhalb der Docker Umgebung. 4. Der Benutzer erreicht die Datenbank nicht mehr.
Mögliche Erweiterungen:	Der Angriff und die Abwehrmassnahme sind auf der Angreifer App dokumentiert.

Use Case Nummer:	3
Use Case Name:	TLS for Docker Daemon
Primärer Aktor:	Benutzer
Kurzbeschreibung:	Wenn man den Docker-Daemon über einen Netzwerk-Socket exponieren muss, so sollte man die TLS-Authentifizierung konfigurieren. Diese verhindert, dass jeder im Netzwerk auf den Daemon zugreifen kann.
Main Success Scenario:	1. Der Daemon wird über das Netzwerk exponiert. 2. Der Benutzer kann über das Netzwerk auf den Daemon zugreifen. 3. Der Benutzer konfiguriert die TLS-Authentifizierung auf dem Daemon. 4. Der Benutzer kann den Daemon nicht mehr übers Netzwerk erreichen. 5. Der Benutzer richtet das Zertifikat ein. 6. Der Benutzer erreicht den Daemon über das Netzwerk mit TLS-Authentifizierung.
Mögliche Erweiterungen:	Zusätzliche Skripts, welche die TLS-Konfiguration ausführen. So muss der Benutzer nicht alles manuell machen.

Use Case Nummer:	4
Use Case Name:	Denial of Service (DoS)
Primärer Aktor:	Benutzer & Angreifer App
Kurzbeschreibung:	Standardmässig haben alle Container Zugriff auf die volle Rechenleistung der Infrastruktur. Wenn ein Container also die gesamte Rechenleistung blockiert, so können die anderen Container nicht mehr arbeiten.
Main Success Scenario:	1. Alle Container verfügen über die gesamte Rechenleistung. 2. Der Benutzer startet einen DoS-Angriff von der Angreifer App. 3. Die ExpressCart Anwendung ist nicht mehr verfügbar. 4. Der Benutzer begrenzt die Rechenleistung der Container. 5. Der Benutzer startet einen DoS-Angriff von der Angreifer App. 6. Die ExpressCart Anwendung bleibt verfügbar.
Mögliche Erweiterungen:	- Der DoS-Angriff und die Abwehrmassnahmen sind auf der Angreifer App dokumentiert. - Die CPU-Auslastung wird während des DoS-Angriffs visualisiert (Zusammenhang mit Container Outbreak). - Ein DoS-Angiff auf das Netzwerk ausführen.

Use Case Nummer:	5
Use Case Name:	Secret in Dockerfile
Primärer Aktor:	Benutzer
Kurzbeschreibung:	Geheime Passwörter sollten niemals im Dockerfile abgelegt werden. Das Dockerfile wird oft zusammen mit dem Code in einem öffentlichen Repository gespeichert. Somit könnte jeder das Passwort auslesen.
Main Success Scenario:	1. Dockerfile enthält ein Secret als Argument. 2. Der Benutzer sieht das Secret beim Build des Images. 3. Das Secret wird in «Docker Secrets» verwaltet und ist somit nicht mehr im Dockerfile.
Mögliche Erweiterungen:	Die Datenbank könnte mit einem Passwort geschützt werden und dieses in Docker Secrets verwaltet werden.
Use Case Nummer:	6
Use Case Name:	Falsches Image
Primärer Aktor:	Benutzer
Kurzbeschreibung:	Das Image sollte immer explizit gewählt werden. Wenn das Image z.B. mit «latest» ausgewählt wird, so hat man keine Kontrolle, welche exakte Version geladen wird.
Main Success Scenario:	1. Der Host besitzt zwei Docker Images (neu und alt) der ExpressCart App. 2. Der Benutzer startet den Container mit dem «latest»-Tag für das Image. 3. Im Container läuft das ältere Image.
Mögliche Erweiterungen:	

Use Case Nummer:	7
Use Case Name:	Container Outbreak
Primärer Aktor:	Angreifer App
Kurzbeschreibung:	Wenn ein Docker Container im privileged Modus betrieben wird, so hat er uneingeschränkten Zugriff auf alle Ressourcen des Betriebssystems.
Main Success Scenario:	1. Die Angreifer App läuft im privileged Modus. 2. Der Benutzer kann über die App einen CLI-Command ausführen. 3. Die Angreifer App wird ohne das «Privileged»-Flag gestartet. 4. Die Angreifer App kann nicht mehr auf die CLI zugreifen.
Mögliche Erweiterungen:	Im Zusammenhang mit Use Case 4 (Denial of Service) könnte die CPU-Last gemessen werden.

Use Case Nummer:	8
Use Case Name:	XSS Angriff
Primärer Aktor:	Benutzer & Angreifer App
Kurzbeschreibung:	Wenn Inputs nicht validiert werden, so kann ein Angreifer ein JavaScript in die Datenbank einschleusen. Sobald ein Benutzer diese Daten aus der Datenbank lädt, so wird der JavaScript Code auf seinem Rechner ausgeführt.
Main Success Scenario:	1. Die Angreifer App schleust ein JavaScript in die Datenbank ein. 2. Die Angreifer App zeigt dem Benutzer ein Link, womit das JavaScript aus der Datenbank geladen wird. 3. Das JavaScript wird auf dem Rechner des Benutzers ausgeführt.
Mögliche Erweiterungen:	Die Abwehrmethoden implementieren, so dass kein XSS-Angriff mehr ausgeführt werden kann.

Use Case Nummer:	9
Use Case Name:	XSRF Angriff
Primärer Aktor:	Benutzer & Angreifer App
Kurzbeschreibung:	Ist das Opfer in ExpressCart angemeldet und hat ein Cookie, so kann der Angreifer einen vorgefertigten Link auf seiner Website haben, welcher z.B. einen Kauf tätigt sobald das Opfer die Seite aufruft.
Main Success Scenario:	1. Der Benutzer meldet sich bei ExpressCart an. 2. Der Benutzer öffnet die XSRF-Seite der Angreifer App. 3. Der Cross-Site-Request fügt einen Artikel zum Warenkorb hinzu.
Mögliche Erweiterungen:	Die Abwehrmethoden implementieren, so dass kein XSRF-Angriff mehr ausgeführt werden kann.

Use Case Nummer:	99
Use Case Name:	Virtual Box Image erstellen
Primärer Aktor:	
Kurzbeschreibung:	Die Applikationen sollen in einem vorgefertigten Virtual Box Image laufen. Dies vereinfacht die Nutzung der gesamten Demosoftware.
Main Success Scenario:	1. Es wird ein Virtual Box Image erstellt, welches alle Szenarien beinhaltet. 2. Die virtuelle Maschine kann über das Netzwerk vom Host erreicht werden.
Mögliche Erweiterungen:	

2.4 Threats

2.4.1 Public Access (CIS 5.13)

Einleitung

Laut den Standardeinstellungen können Docker-Container Verbindungen nach aussen aufbauen, jedoch nicht von aussen erreicht werden. Damit ein Container von der Aussenwelt erreicht werden kann, muss ein Netzwerk-Socket dafür freigegeben werden.

Wird ein Container normal gestartet und dabei lediglich ein Port freigegeben, so wird der Container automatisch an das Wildcard-Interface 0.0.0.0 gebunden. Somit ist der Container auf sämtlichen Interfaces zu erreichen. Wenn eine Applikation jedoch in einer geschützten Umgebung laufen soll und somit über das Netzwerk nicht erreichbar sein darf, so stellt dies ein Problem dar.

Angriff

Bei diesem Angriff muss der Angreifer die Konfiguration der Docker-Umgebung kennen, respektive erraten. Er versucht, Applikationen auf Ports zu erreichen, welche nicht extern erreichbar sein sollten. Dies kann er entweder mit einem Portscan machen, sofern dieser nicht unterdrückt wird, oder er kann bekannte, standardisierte Ports gezielt abfragen.

Abwehr

Damit eine Container-Applikation nicht von jedem Interface aus erreicht werden kann, sollte bei jeder Portfreigabe das Interface explizit angegeben werden.

Schlecht:	Empfehlung:
`docker run -p 1234:1234`	`docker run -p 10.0.0.8:1234:1234`
`ports:` `  - 1234:1234`	`ports:` `  - 10.0.0.8:1234:1234`

Integration in Demo-Software

ExpressCart verwendet zur Speicherung der Daten eine MongoDB. Diese läuft in einem eigenen Container, welcher auf dem MongoDB-Standartport 27017 erreicht werden kann. Da bei der Freigabe des Ports kein Interface definiert wurde, ist die Datenbank auf allen Interfaces erreichbar. Somit kann jeder im Netzwerk die Datenbank auf Port 27107 erreichen und Aktionen darauf ausführen.

Da wir in der Demo-Software nicht mehrere Interfaces verwenden, definieren wir den «localhost» als internes Interface. Wenn nun also im «docker-compose.yml» die Applikation an den Socket `127.0.0.1:27017:27017` gebunden wird, so ist diese nur noch auf dem localhost zu erreichen und nicht mehr von extern.

Überprüfung

`docker container ls` zeigt alle laufenden Docker-Container, inklusive der Port-Bindings.

Abbildung 8: Screenshot des Befehls "docker container ls"

Die Markierung in der obigen Abbildung zeigt die Port Bindings an. Hier sollte ein explizites Interface definiert sein.

2.4.2 Unrestricted Traffic between Containers (CIS 2.1)

Einleitung

Standardmässig gibt es innerhalb eines Docker-Netzwerks keine Zugriffsbeschränkungen zwischen den Containern. Wenn ein Container direkt über Docker gestartet (`docker run …`) wird, so läuft dieser standardmässig im Bridge-Netzwerk. Für alle Container, welche über Docker-Compose gestartet werden, wird ein neues Netzwerk generiert und die Container befinden sich gemeinsam in diesem Netzwerk.

Angriff

Ein «angreifender» Docker-Container muss in dasselbe Netzwerk kommen, wie der anzugreifende Container. Befindet sich der anzugreifende Container lediglich im Bridge-Netzwerk, so kann der anzugreifende Container normal über Docker gestartet werden.

Befindet sich der anzugreifende Container jedoch in einem separierten Netzwerk, so muss der Angreifer ebenfalls in dieses Netzwerk kommen.

Abwehr

Im Host-Netzwerk sollten niemals Container laufen, welche untereinander kommunizieren müssen. Deshalb kann «Inter Container Traffic» im Bridge-Netzwerk gänzlich unterdrückt werden. Um diese Einstellung zu aktivieren muss Docker zuerst gestoppt werden.

```
sudo systemctl stop docker
dockerd --icc=false
sudo systemctl start docker
```

Ansonsten können Netzwerke nicht weiter geschützt werden. Um zu verhindern, dass Netzwerknamen von Angreifern erraten werden, könnte man diese randomisieren. Dies ist aber kein Zugriffsschutz.

Integration in Demo-Software

In der Demo-Software wird versucht, von der Angreifer-App aus auf die Daten in der MongoDB zuzugreifen. Da die beiden Container standardmässig in zwei verschiedenen Netzwerken laufen, muss der Angreifer zuerst in das Netzwerk der ExpressCart-App gebracht werden. Danach kann die Angreifer-App die Daten aus der MongoDB auslesen.

Der einzige sichere Schutz ist hier eine Passwort-Authentisierung. Da diese für die Übung mit den «Secrets» benötigt wird, soll der Anwender, den Datenbankzugriff mit einem Passwort schützen.

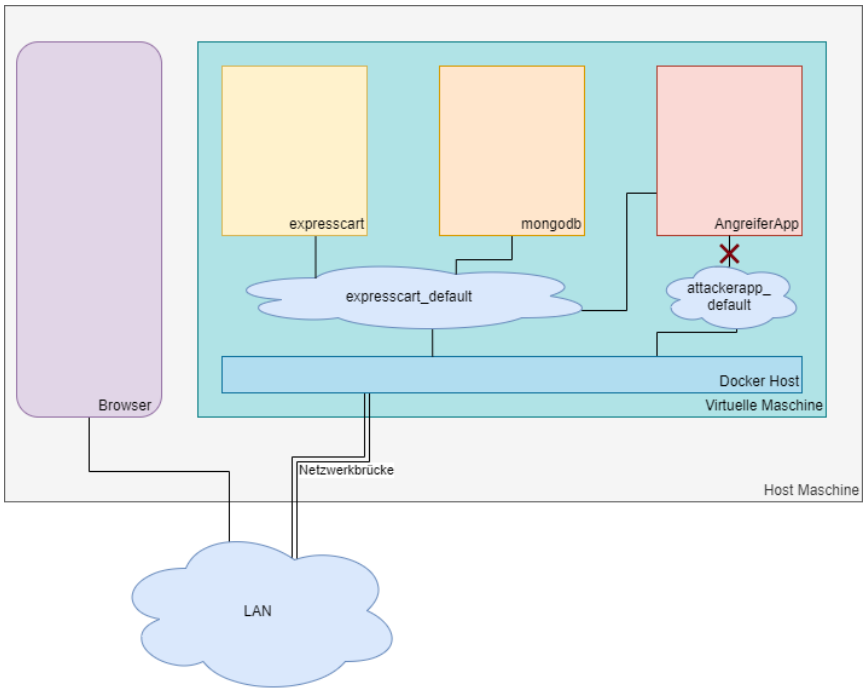

Abbildung 9: Übersichtsschema nach Anpassung des Netzwerks

Überprüfung

- Auflisten aller Netzwerke: `sudo docker network ls`
- Netzwerk überprüfen: `sudo docker network inspect <network-id>`

Der Eintrag `"com.docker.network.bridge.enable_icc": "false"` zeigt an, dass kein «Inter Container Traffic» in diesem Netzwerk erlaubt ist.

Abbildung 10: Ausschnitt aus einem Network Inspect

Beim Netzwerk des obigen Ausschnitts ist Inter Container Traffic erlaubt, anderenfalls müsste beim Pfeil «false» stehen.

2.4.3 TLS for Docker Daemon (CIS 2.6)

Einleitung

Bei Docker besteht die Möglichkeit, die Konfiguration der Docker Engine über eine Web-API anzupassen. Damit nicht jeder, der den Pfad der API kennt, Anpassungen der Docker Engine machen kann, ist es dringenst empfohlen, den API-Zugriff mit einem TLS-Zertifikat abzusichern, sobald die Freigabe auf dem Netzwerk erfolgt.

Angriff

Falls sich ein ungeschützter Daemon-Socket in einem Netzwerk (im schlimmsten Fall auch das Internet gemeint) befindet, welches der Angreifer erreichen kann, so kann dieser ungehindert Veränderungen an Docker vornehmen. Der Berechtigungsgrad der API ist äquivalent zu «root». Somit hat der Angreifer uneingeschränkten Zugriff auf den Daemon und somit auf das Host-System. Wenn man also den Docker Daemon auf dem Netzwerkinterface freigibt und noch keine TLS-Zertifizierung eingerichtet hat, so kann der Angreifer jegliche Konfigurationen auf dem Daemon vornehmen.

Abwehr

Als Grundsatz gilt, wenn immer möglich den Daemon nicht ans Netzwerk zu binden. Somit ist es auch nicht nötig diesen zu schützen.

Falls es doch nötig ist, den Daemon übers Netzwerk zu erreichen, so wird dringendst empfohlen diesen mit TLS zu schützen. Dabei stellt das Host-System Zertifikate aus, welche manuell auf den Benutzer-Clients installiert werden müssen. Bei jedem Zugriff wird überprüft, ob ein gültiges Zertifikat mitgeschickt wurde. Damit werden unautorisierte Zugriffe verhindert.

Das Benutzerzertifikat sollte mit der gleichen Vertraulichkeit behandelt werden, wie das root-Passwort. Denn wie bereits erwähnt, hat man root-Rechte bei einem Netzwerkzugriff.

Integration in Demo-Software

In der Demo-Umgebung wird zuerst gezeigt, wie der uneingeschränkte Netzwerkzugriff auf den Daemon funktioniert, indem man diesen an einen Netzwerk-Port bindet. Im Anschluss kann man von extern auf den Daemon zugreifen und zum Beispiel die Version abfragen.

Nun sollen die Zertifikate mittels «OpenSSL»[5], gemäss offizieller Anleitung[6], eingerichtet werden. Anschliessend muss der Daemon mit der Zertifizierungs-Bedingung neu gestartet werden.

Der Netzwerkzugriff funktioniert nun nur noch, wenn die Zertifikate auch mitgeliefert werden. Das Mitschicken der OpenSSL-Zertifikate funktioniert, mit dem Standard-«cURL»-Befehl, nur auf Unix Maschinen.

[5] OpenSSL: https://www.openssl.org/ [Abrufdatum: 13.11.20]
[6] Docker docs: https://docs.docker.com/engine/security/https/ [Abrufdatum: 13.11.20]

Überprüfung

Die Startup-Optionen des Docker Daemons können mit folgendem Befehl überprüft werden (Center for Internet Security, 2019).

```
sudo ps -ef | grep dockerd
```

Hier ist es wichtig, dass die folgenden Flags gesetzt sind, um die TLS-Verifizierung sicher zu stellen.

```
--tlsverify
--tlscacert
--tlscert
--tlskey
```

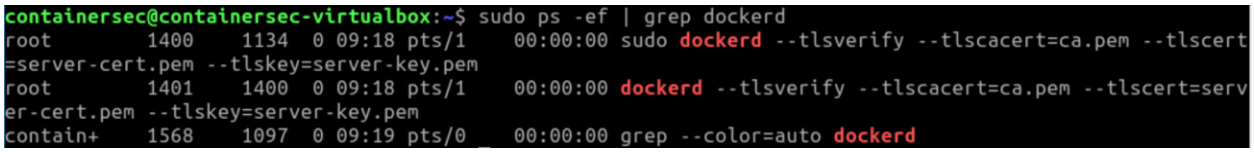

Abbildung 11: Überprüfung der Startup-Optionen vom Docker Daemon

2.4.4 Container Outbreak (CIS 4.1)

Einleitung

Die Containerisierung von Applikationen hat unter anderem zum Ziel, die Applikation zu einem gewissen Grad vom Host System zu trennen, um das Sicherheitsmanagement zu verwalten. Damit man vom Container aus, das Host System nicht angreifen kann, müssen bei der Initialisierung einige Punkte beachtet werden.

Angriff

Standardmässig wird ein Container vom «root» Benutzer gestartet. Dies hat zur Folge, dass der Container auch root-Rechte auf dem Host System besitzt. Wenn man es also schafft, Befehle im Container auszuführen, so besteht die Möglichkeit auch Änderungen am Host-System vorzunehmen.

Welche Änderungen genau vorgenommen werden können, hängt wiederum vom Betriebssystem und der Docker-Konfiguration ab. Wenn man Zugriff auf das Dateisystem des Hosts hat, kann man diesen vom Container aus komplett umkonfigurieren.

Abwehr

Bei den Volumes sollten «bind mounts» möglichst vermieden werden. Falls es ein «bind mount» benötigt, sollte dieses so restriktive wie möglich gehalten werden. So verhindert man, dass der Container auf Dateien des Hosts zugreifen kann.

Des Weiteren sollte der Container niemals unter dem «root»-Benutzer laufen. Deshalb sollte bei jedem Start eines Containers ein Benutzer explizit angegeben werden. Dies ist einer der wichtigsten Sicherheits-Vorkehrungen, ansonsten findet keine Kapselung mehr statt.

Integration in Demo-Software

Bei dieser Aufgabe gehen wir davon aus, dass die Angreifer-App Lesezugriff auf gewisse Host Ressourcen benötigt. Der Einfachheit halber wurde deshalb der `/etc` Ordner des Host-Systems dem Conatiner hinzugefügt.

Als erstes soll nun der Anwender den Container wie gewohnt, ohne spezifischen Benutzer, starten. In der Angreifer-App befindet sich nun eine Schaltfläche, welche über einen Konsolen-Befehl den Inhalt der «sudoers»-Datei ausgibt. Für diese Datei hat ausschliesslich der «root» leserechte.

Nun wird im Dockerfile ein neuer Benutzer erstellt und der Container unter diesem Benutzer neu gestartet. Versucht der Anwender nun wieder die «sudoers»-Datei auszulesen erhält er ein «Permission denied».

Überprüfung

Mit dem folgenden Command kann überprüft werden, welche Container unter welchen Benutzern laufen.

```
docker inspect $(docker image ls -q) --format '{{.Config.User}} {{.Name}}'
```

Hier ist es wichtig, dass jeder Container mit einem Benutzer läuft.

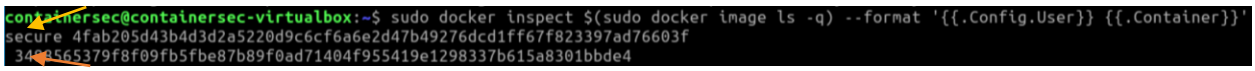

Abbildung 12: Überprüfung der Container-Benutzer

Der obere Pfeil zeigt an, dass dieser Container vom Benutzer «secure» ausgeführt wird. Dem unteren Container ist kein Benutzer zugewiesen, was bedeutet, dass dieser standardmässig vom «root» Benutzer ausgeführt wird.

2.4.5 Denial of Service (CIS 5.10 & 5.11)

Einleitung

Virtualisierungen werden verwendet, um Hardware zu teilen und somit effizienter zu nutzen. Im Gegensatz zu virtuellen Maschinen, wo die System-Ressourcen in fixierte reservierte Partitionen unterteilt wird, wird bei Containern die unterliegende System-Ressourcen standardmässig dynamisch geteilt. Dies reduziert den Overhead, da man nicht für jeden Container eine Reserve einplanen muss, sondern nur für das Gesamtsystem. Für dieses Kapitel gilt jeweils die Voraussetzung, dass mehrere Container auf einem Host laufen.

Angriff

Die Systemressourcen werden üblicherweise aufgeteilt in die Kategorien: Rechenleistung, Arbeitsspeicher und Festplattenspeicher. Die Idee hinter dem Angriff ist in allen Kategorien jedoch dieselbe. Wenn ein Container eine Ressource zu 100 Prozent auslastet, so bleibt für die anderen Container nichts mehr übrig.

Wenn ein Angreifer also eine «sichere»[7] Applikation lahmlegen möchte, so kann er dies machen, indem er eine «unsichere»[8] Applikation dazu bringt, die gesamten System-Ressourcen für sich zu beanspruchen.

Der Festplattenspeicher kann in den neusten[9] Docker-Umgebungen noch nicht beschränkt werden. Wenn man den Festplattenspeicher beschränken möchte, so muss man ein Volume mit einer festen Grösse erstellen. Standardmässig hat ein Volume 10GB.

Abwehr

Für jeden Docker Container besteht die Möglichkeit, die Rechenleistung und den Arbeitsspeicher zu beschränken. Es kann angegeben werden, wie viele CPUs (Bsp. 0.5 für 50% maximale Auslastung bei einer verfügbaren CPU) oder wie viel Memory, in Bytes, der Container maximal verwenden darf.

Integration in Demo-Software

In der Demo-Umgebung überwacht der Benutzer die CPU-Auslastung der virtuellen Host Maschine. In der Angreifer-App kann man einen Service starten, welcher die CPU für 15 Sekunden auslastet, indem er komplexe Multiplikationen durchführt. Hier sollte während dieser Zeit eine CPU-Auslastung gegen 100 Prozent gemessen werden können.

Im Anschluss wird die maximale CPU-Auslastung für diesen Container auf 50 Prozent beschränkt und der Service nochmals gestartet. Nun sollte die CPU-Auslastung nicht markant über 50 Prozent gehen. Der Container sollte bei 50 Prozent beschränkt sein, dazu kommen noch die restlichen Host-Prozesse, welche auch noch etwas CPU benötigen.

[7] Applikation, welche nicht angreifbar ist.
[8] Verwundbare Applikation.
[9] Stand: 01.12.2020, Version: 19.03.13

Überprüfung

Mit dem Command `docker container inspect <container-id>` kann man die Konfiguration eines Docker Containers betrachten. Wenn man nach dem Stichwort `NanoCpus` sucht, so sollte man die maximal verfügbare CPU-Leistung sehen. Hier muss berücksichtigt werden, dass die Angabe in nano CPUs gemacht wird und somit den Faktor 10^{-9} trägt. Ein NanoCPU-Wert von `0` bedeutet, dass keine CPU-Beschränkung besteht.

Die Abbildung auf der rechten Seite zeigt einen Wert von 500'000'000 NanoCpus an. Multipliziert mit dem Faktor 10^{-9} ergibt dies die Beschränkung auf 0.5 Cpus. Somit kann die CPU nicht über 50 Prozent belastet werden.

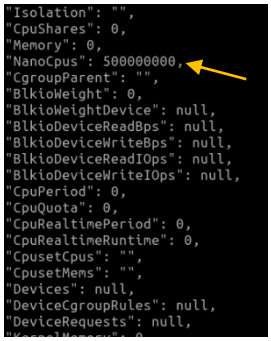

Abbildung 13: Überprüfung CPU-Limit

2.4.6 Secret in Dockerfile (CIS 4.10)

Einleitung

Bei vielen Applikationen ist es nötig eine Passwort-Authentifizierung durführen zu können, zum Beispiel für einen Datenbankzugriff. Dieses Passwort sollte keinesfalls direkt in den Source-Code programmiert werden, da es so mittels Reverse Engineering geknackt werden könnte. Docker bietet dafür ein spezielles Secrets Management an.

Angriff

Wenn geheime Schlüssel direkt in den Source Code programmiert werden, so kann es sein, dass diese mittels reverse Engineering herausgefunden werden können. Werden die Passwörter direkt in die Umgebungsvariablen eingegeben, so werden sie ebenfalls in Klartext zwischengespeichert, während dem der Container läuft.

Abwehr

Docker bietet spezielle Tools an, um Passwörter zu verwalten. So wird ein Passwort nicht als Klartext gehandhabt, sondern in einem File gespeichert. Zudem vereinfacht eine solche homogene Passwortverwaltung Änderungen der Passwörter.

Integration in Demo-Software

Die Datenbankverbindung von ExpressCart wurde, als Abwehrmassnahme bei Kapitel 3.3.2, Unrestricted Traffic between Containers, mit einer Authentifizierung versehen. Das Passwort dieser Verbindung war bis anhin in Klartext im Dockerfile geschrieben. Das Passwort soll nun in ein externes File ausgelagert werden. Mittels «Secrets» wird dieses Passwort dann ins «docker-compose.yml» eingebunden.

Überprüfung

Dieses Sicherheits-Feature kann nicht mit einem Konsolenbefehl überprüft werden. Hier ist eine gute Abstimmung zwischen Entwicklung und Deployment nötig. Es muss schon bei der Entwicklung darauf geachtet werden, dass die Passwörter in Dateien übergeben werden.

2.4.7 Falsches Image (CIS 5.27)

Einleitung

Im Docker Hub[10] stehen verschiedenste Images zur Verfügung, welche für die Grundlage der eigenen Images verwendet werden können. Die Images identifizieren sich jeweils mit einem Namen und einem Tag, welcher die «Version» angibt.

Angriff

Ein möglicher Angriff wäre, ein Image zu verändern und eine Schwachstelle einzubauen. Somit hat jeder, welcher dieses Image verwendet diese Schwachstelle im System. Da die Repositorien in Docker Hub aber relativ gut überwacht sind, ist dieser Angriff recht unwahrscheinlich. Jedoch könnte es vorkommen, dass man alte Images mit bekannten Schwachstellen verwendet, wenn man die Tags falsch handhabt.

Abwehr

Jedes Image sollte immer mit einem spezifischen Tag angegeben werden. Es gibt die Möglichkeit, keinen Tag anzugeben, dann wird automatisch der «latest»-Tag verwendet. Davon ist aber gänzlich abzuraten, da das «latest»-Image meist nicht das aktuellste ist und es ebenfalls getagt wird, wenn jemand beim push keinen Tag angibt. Dies ist meist kein gutes Zeichen.

Wenn man sicher gehen will, dass ein bestimmtes Image unverändert geladen wird, so kann anstatt des Tags, der Hash des Images angeben. Somit ist die Integrität gewährleistet.

Integration in Demo-Software

In der Angreifer-App wird das Image «node:12» verwendet. Nun soll ein neues Image geladen werden, welches mit dem Hash angegeben wird.

Zudem soll man sich einen Überblick auf Docker Hub verschaffen. Hier wird ein besonderes Augenmerk daraufgelegt, dass bei vielen Image-Sammlungen das «latest»-Image nicht das Neuste ist.

Überprüfung

Mit dem Command `docker ps -a` werden alle verfügbaren Container aufgelistet. In der Spalte «Image» ist das Basisimage zu sehen. Hier ist empfohlen, dass mindestens ein spezifischer Tag gesetzt ist.

[10] Docker Hub: https://hub.docker.com/

2.5 Automated Container Auditing

Um Container auf Sicherheitsrisiken zu überprüfen, gibt es einige Tools. Hier wird das Tool «Docker Bench for Security»[11] analysiert. Das Tool prüft die laufenden Container mit verschiedenen Tests. Es gibt gewertete und nicht gewertete Tests. Als Resultat gibt es ein Score. Dieser gibt an, wie viele gewertete Tests bestanden wurden. Bei den nicht gewerteten Tests erfolgt jeweils einfach eine Information, da diese entweder schwer zu testen oder nicht sicherheitsrelevant sind.

```
# ------------------------------------------------------------------------
# Docker Bench for Security v1.3.5
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker containers in production.
# Inspired by the CIS Docker Benchmark v1.2.0.
# ------------------------------------------------------------------------

Initializing Tue Nov  5 10:27:42 UTC 2019


[INFO] 1 - Host Configuration

[INFO] 1.1 - General Configuration
[NOTE] 1.1.1  - Ensure the container host has been Hardened
[INFO] 1.1.2  - Ensure Docker is up to date
[INFO]        * Using 19.03.4, verify is it up to date as deemed necessary
[INFO]        * Your operating system vendor may provide support and security maintenance for Docker

[INFO] 1.2 - Linux Hosts Specific Configuration
[WARN] 1.2.1 - Ensure a separate partition for containers has been created
[INFO] 1.2.2  - Ensure only trusted users are allowed to control Docker daemon
[INFO]        * docker:x:998:
[WARN] 1.2.3  - Ensure auditing is configured for the Docker daemon
[WARN] 1.2.4  - Ensure auditing is configured for Docker files and directories - /var/lib/docker
[WARN] 1.2.5  - Ensure auditing is configured for Docker files and directories - /etc/docker
[WARN] 1.2.6  - Ensure auditing is configured for Docker files and directories - docker.service
[WARN] 1.2.7  - Ensure auditing is configured for Docker files and directories - docker.socket
[WARN] 1.2.8  - Ensure auditing is configured for Docker files and directories - /etc/default/docker
[INFO] 1.2.9  - Ensure auditing is configured for Docker files and directories - /etc/sysconfig/docker
[INFO]        * File not found
[INFO] 1.2.10  - Ensure auditing is configured for Docker files and directories - /etc/docker/daemon.json
[INFO]         * File not found
[WARN] 1.2.11  - Ensure auditing is configured for Docker files and directories - /usr/bin/containerd
[INFO] 1.2.12  - Ensure auditing is configured for Docker files and directories - /usr/sbin/runc
[INFO]         * File not found


[INFO] 2 - Docker daemon configuration
[WARN] 2.1  - Ensure network traffic is restricted between containers on the default bridge
[PASS] 2.2  - Ensure the logging level is set to 'info'
[PASS] 2.3  - Ensure Docker is allowed to make changes to iptables
```

Abbildung 14: Beispiel einer "Docker Bench for Security" Auswertung, Quelle: (Docker, 2020)

[11] Docker Bench for Security: https://github.com/docker/docker-bench-security [04.12.2020]

2.5.1 Audit der Demonstrationsumgebung

Es wurden zwei Audits der Demonstrationsumgebung durchgeführt. Das erste Audit wurde gemacht, als die beiden Applikationen ohne die Sicherheitsvorkehrungen in Docker liefen. Dabei erzielten die Container gemeinsam einen Score von 18 Punkten.

```
[PASS] 7.9  - Ensure CA certificates are rotated as appropriate (Swarm mode not
enabled)
[PASS] 7.10  - Ensure management plane traffic has been separated from data plan
e traffic (Swarm mode not enabled)

[INFO] Checks: 105
[INFO] Score: 18
containersec@containersec-virtualbox:~$
```

Abbildung 15: Auswertung ohne spezielle Sicherheitsvorkehrungen

Während der Übung in der Demonstrationsumgebung werden, in Bezug auf Docker, die folgenden Sicherheitslücken geschlossen:

- Die Datenbank wird an ein fixes Interface gebunden
- Der Docker Daemon Zugriff wird mit einem TLS-Zertifikat geschützt
- Die Angreifer App läuft unter einem eigenen Benutzer
- Die Angreifer App kann die CPU nicht mehr als 50 Prozent auslasten
- Für das Datenbankpasswort wird ein Secret erstellt
- Das Node Image der Angreifer App wird mit dem Hash überprüft

Danach wurde das zweite Security Audit durchgeführt. Dieses ergab ebenfalls einen Score von 18 Punkten.

```
t enabled)
[PASS] 7.9  - Ensure CA certificates are rotated as appropriate (Swarm mode not
enabled)
[PASS] 7.10  - Ensure management plane traffic has been separated from data plan
e traffic (Swarm mode not enabled)

[INFO] Checks: 105
[INFO] Score: 18
containersec@containersec-virtualbox:~$
```

Abbildung 16: Auswertung mit den getroffenen Sicherheitsvorkehrungen

Hier hat man eine Applikation, welche einige der zentralen Sicherheits Best Practices umgesetzt hat. Trotzdem erzielt diese keinen besseren Score als eine Applikation, welche keine Sicherheitsrichtlinien befolgt. Der Grund, dass beide Ergebnisse gleich sind, liegt an dem Messverfahren dieses Tools. Damit ein Punkt als bestanden gilt, muss der Test auf allen drei Containern bestanden sein.

Das Ergebnis eines solchen Tests muss daher immer mit Vorsicht betrachtet werden. Zum einen haben unterschiedliche Applikationen auch unterschiedliche Anforderungen an die Sicherheit, zum anderen gibt es noch zahlreiche nicht gewertete Punkte, welche sicherheitsrelevant sind. Zudem werden die Container mit 105 gleich gewichteten Tests geprüft, was zu unübersichtlichen Resultaten führt.

2.6 Fazit

Das Ziel dieser Arbeit war die Entwicklung einer Software für die Veranschaulichung von Docker Security Best Practices. Es resultierte dabei eine Applikation, mit welcher sieben Vulnerabilitäten nachgestellt werden können. Während der Arbeit hat sich herausgestellt, dass Docker nicht nur die Vorteile der Flexibilität gegenüber virtuellen Maschinen mit sich bringt, sondern auch einige Sicherheitsaspekte beachtet werden müssen. Hier empfiehlt es sich eigene Standards zu definieren. Zudem stellte sich heraus, dass statische Analyse Tools wie «Docker Bench for Security» nur bedingt für die Überprüfung geeignet sind.

Für die Erarbeitung von Docker Sicherheitsstandards ist es empfehlenswert, Dokumente, wie die «CIS Docker Benchmark» oder die «OWASP Docker Top 10» beizuziehen. Es ist aber wichtig, dass die Standards immer mit der Umgebung abgestimmt werden. Jede Applikation bringt ihre eigenen Anforderungen und Tücken in Bezug auf die Sicherheit.
Die automatisierten Tools können helfen, einzelne Punkte automatisch zu überprüfen, der gesamt Score hat jedoch wenig Aussagekraft. Dies liegt daran, dass die Tests nicht gewichtet sind. Momentan wird die Aktivierung eines Logs gleich wie ein privilegierter Container gewichtet. Dies kann die Resultate stark verfälschen.

Dieser Bericht beschränkt sich auf sieben Sicherheitsaspekte, welche anhand von Recherche als allgemein relevant eingestuft wurden. In der Zukunft wäre es möglich, die Software weiter auszubauen, entweder mit neuen Vulnerabilitäten oder die bestehende Software zu verbessern, indem man die einzelnen Aufgaben weiter ausführt. Bei den automatisierten Auditing Tools wäre es hilfreich, wenn man eine Gewichtung festlegen könnte. So würde der Score auch aussagekräftig. Zudem sollten die Container einzeln überprüft und gewertet werden können. Bei Docker wäre es hilfreich, wenn man für die Erstellung und Verwendung eines Users nur eine Zeile Code schreiben müsste und nicht drei bis vier wie bis anhin.

3 Glossar

Angreifer App	Die Angreifer App simuliert eine böse Applikation in der Demonstrationsumgebung.
API	Ein API ist eine Programmierschnittstelle.
Attack-Tree	Ein Attack-Tree ist eine visuelle Hilfe, um Angriffsmöglichkeiten logisch zu verknüpfen.
CSS-Stylesheet	CSS ist eine Programmiersprache, mit welcher das Design von Webseiten programmiert wird. Das Stylesheet enthält den Code.
Economy of Scale	Senkung der durchschnittlichen Produktionskosten und damit der Stückkosten, wenn der Output erhöht wird (Oxford Languages).
ExpressCart	ExpressCart ist eine Open Source Webshop Applikation. Sie simuliert die verwundbare Applikation in der Demonstrationsumgebung.
Gesamtscore	Der Gesamtscore ist das zusammengefasste Resultat mehrerer Tests, welches in einer Zahl ausgedrückt wird.
ISO	Internationale Organisation für Normung
MongoDB	Ist ein Datenbankmanagementsystem, welches in ExpressCart verwendet wird.
Node.js	Programmiersprache, welche für die Programmierung von Webservern entwickelt wurde.
Overhead	Zusätzliche Ressourcen, welche benötigt, aber meist nicht genutzt werden.
Threat-Model	Ein Prozess, mit dem potenzielle Bedrohungen identifiziert und aufgezählt werden können (Wikipedia).
Use Cases	Beschreibungen von konkreten Anwendungen einer Software aus Benutzersicht.
Vulnerabilitäten	Synonym für Verwundbarkeit, oft verwendet im Zusammenhang mit IT-Security.
XSRF	Cross Site Request Forgery. Angriff, bei dem der Angreifer eine Transaktion durchführt im Namen des Opfers.
XSS	Cross Site Scripting. Angriff, bei dem schadenstiftende Daten in ein System eingeschleust werden.

4 Literaturverzeichnis

Center for Internet Security. 2019. CIS Docker Benchmark . 29. 07 2019. S. 52.

Docker. Docker Security-Banner-01. [Online] [Zitat vom: 09. 12 2020.] https://www.docker.com/blog/docker-engine-1-10-security/docker-security-banner-01/.

—. 2020. docker-bench-security. *Readme.md.* [Online] 30. 11 2020. [Zitat vom: 04. 12 2020.] https://github.com/docker/docker-bench-security/blob/master/README.md.

OpenSSL. OpenSSL. [Online] [Zitat vom: 13. 11 2020.] https://www.openssl.org/.

OWASP. 2019. Docker-Security Threats. [Online] 19. 06 2019. [Zitat vom: 02. 12 2020.] https://github.com/OWASP/Docker-Security/raw/master/assets/threats.png.

OWASP Foundation. 2020. OWASP Homepage. [Online] 2020. [Zitat vom: 02. 12 2020.] https://owasp.org/.

Oxford Languages. Oxford Reference. [Online] [Zitat vom: 11. 12 2020.] https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095741513#:~:text=Reductions%20in%20the%20average%20cost,costs%2C%20when%20output%20is%20increased.&text=Internal%20economies%20of%20scale%20occur,financing%2C%20and%20development%2C%20etc..

Sysdig. 2019. *2019 Container Usage Report.* 2019.

Wikipedia. Threat model. [Online] [Zitat vom: 11. 12 2020.] https://en.wikipedia.org/wiki/Threat_model#:~:text=Threat%20modeling%20is%20a%20process,and%20mitigations%20can%20be%20prioritized..

5 Abbildungsverzeichnis

6 Anhang

6.1 Anhang 1 Eigenständigkeitserklärung

OST Ostschweizer Fachhochschule

Eigenständigkeitserklärung

Erklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selbst und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- dass ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum:
Jona, 11.12.20

Name, Unterschrift:

F. Caspani

Fabio Caspani

6.2 Anhang 2 Factsheet zum VDI Image mit Demo-Software

Factsheet ContainerSecurity.vdi

OS: Lubuntu 20.04.1

```
user: containersec
password: password
```

Make sure the Network is in **Bridged Mode**. If there is a problem with DHCP in Bridged Mode, you can configure a static IP for either your VM and your Host Machine.

Installed Packages:

- OpenSSH-Server
    - It's possible to connect to the VM with SSH.
- Docker
- Docker Compose
- Container Security Software in the Home Directory
- Gnome-System-Monitor

6.3 Anhang 3 Aufgabenstellungen der Demo-Software

ContainerSecurity

Table of Content

Description

This is a sample application to simulate some Security Issues in the context of Docker. It bases on the Open Source web shop ExpressCart as a vulnerable sample application. In addition, there is a AttackerApp which simulates the threats.

You will solve several tasks where you will attack the ExpressCart app. The attacks will be launched either from your host or from the AttackerApp. Afterwards you will protect the ExpressCart app. Thus, you will take on the role of both the attacker and the security engineer.

Installation

Virtual Machine

1. Install VirtualBox
2. Download Lubuntu
3. Create a new VM with 1024-2048MB Memory and 10GB Storage
4. Click on the newly created VM and Press `Ctrl + S` to open the settings. In the Network Tab choose *Bridged Adapter*.
   - If you have Problems receiving an IP address on the virtual machine, you can assign a static IP address to the VM. In this Case you must also assign a static IP address to your host computer which is in the same subnet.
5. Start the newly created VM and select the `.iso` File (you downloaded in Step 2) to Boot
6. Select your preferred language
7. Before starting Lubuntu, press `F3` to select your keyboard layout
8. When Lubuntu has started, click `install Lubuntu` and work through the installation wizard

FYI:

- Change Resolution: Start -> Preferences -> LXQt Settings -> Monitor Settings
- Terminal: Start -> System Tools -> QTerminal

Docker

1. Install Docker according to this guide. I recommend to install Docker using the repository.
2. Install Docker-Compose according to this guide. Don't forget to select the Linux Tab.

Software

1. Clone the GitHub Repository ContainerSecurity to the VM.
2. Navigate to `/ContainerSecurity/expressCart` and run `sudo docker-compose up --build`
3. You should now be able to reach ExpressCart from any Host in your Network. Type `<VM-ip>:1111` in your Browser. To get the VM's IP, run the `ip addr` Command.
4. To install the sample data, open another tab in the terminal and run the following:

```
sudo docker exec -it expresscart bash
npm run testdata
exit
```

5. Navigate to `/ContainerSecurity/AttackerApp` and run `sudo docker-compose up --build`
6. You should now be able to reach the AttackerApp from any Host in your Network. Type `<VM-ip>:2222` in your Browser.

Task 1 - Public Access

Table of Content

Introduction

Within the docker environment, all containers are treated equally by default. This means that containers, which are operated in a supposedly "secure" environment, can also be accessed by "public" interfaces.

In this Task, you will access the *expressCart*'s database from the network. Let's assume that you have just started a new business with a small *expressCart* web store you host on your old Computer. To keep the configuration as simple as possible, you just forward all ports to the server.

Access the Database from the Network

1. Run *expressCart* (`sudo docker-compose up --build` in the *expressCart* folder)
2. Make sure *expressCart* is running by accessing the Website (`http://<VM-IP>:1111`) and the sample data is loaded.
3. Get a tool to access MongoDB from your Computer. If you already have a tool (e.g. DataGrip) you can use that, otherwise I would recommend MongoDB Compass.
4. You should now be able to connect and read from the database with the connection string `mongodb://<VM-IP>:27017`.
5. Stop the Docker-Containers.

Remove the public Access

By default, Docker Containers are bound to the wildcard interface `0.0.0.0`, which means they can be reached from any interface. To prevent that, we will manually bind the DB container to the localhost.

1. In the `docker-compose.yml` bind the port including the interface by changing `27017:27017` to `127.0.0.1:27017:27017`.
2. Restart *expressCart* by executing `sudo docker-compose up`
3. Verify that the application works properly but you cannot access the DB any longer.

Task 2 - Inter Container Communication

Table of Content

Introduction

By default, all containers that are in the same network can communicate with each other without restrictions.

In this Task, you will access the *expressCart*'s database from another Container.

Networking in Docker

If you start Docker Containers using Docker Compose, Docker creates a new network for all containers started in the same `docker-compose.yml`. If you just start a container using the `docker run ...` command, the container runs in the default network named *host*.

To prevent inter container communication in the *host* network, you can set the *icc* flag in the Docker Daemon. To do that, you have to stop Docker.

```
sudo systemctl stop docker
dockerd --icc=false
sudo systemctl start docker
```

Prerequisites

First, make sure you are not able to connect to the database from the AttackerApp (**02 ICC --> show DB data**). In order to access the database, you have to deploy the *AttackerApp* in the same network as *ExpressCart*. An existing network can be joined using the following steps:

1. Stop the Docker Container with the AttackerApp.
2. Open the `docker-compose.yml` in the *AttackerApp* folder.
3. Add this Code to the end of the `docker-compose.yml`. It needs to be on the same level as the `services:` (no indent).

```
  networks:
    default:
      external:
        name: expresscart_default
```

4. Restart the Docker Container executing `sudo docker-compose up`

Read the Data using the AttackerApp

1. Open the AttackerApp in your Browser.
2. Navigate to **02 ICC --> show DB data**
3. You should now be able to see all the products stored in the database.

Add Password Protection to mongoDB

To securely protect the database from other containers, it requires a firewall or a password protection. In this case, a password protection is recommended because it is needed in task 6.

1. Add the user/password environment variables to the *docker-compose.yml* of *ExpressCart*.

```
environment:
  - MONGO_INITDB_ROOT_USERNAME=user
  - MONGO_INITDB_ROOT_PASSWORD=password
  - MONGO_INITDB_DATABASE=expresscart
```

2. Change the **databaseConnectionString** in *expressCart/config/settings.json* to `mongodb://<username>:<password>@mongodb:27017/expresscart?authSource=admin`.
3. Unfortunately it is not possible to add a user to an existing database in mongodb, only using Docker Compose. Therefore, the data needs to be deleted and recreated.

```
sudo docker container prune #delete all stopped
Containers
sudo docker volume rm expresscart_expresscart-
mongo-data #delete the Volume
```

4. Afterwards, the Application can be restarted. `sudo docker-compose up --build`
5. Add the Sample Data

```
sudo docker exec -it expresscart bash
npm run testdata
exit
```

6. The DB Connection is now Authenticated.

Task 3 - TLS for Docker Daemon

Table of Content

Introduction

It is possible to expose the Docker Daemon API to the Network. If you do this, it is strongly recommended to secure the connections using TLS. Otherwise, anybody with access to the network can modify your Daemon.

Warning

You have to protect the **certificates** the same way as you would protect the **root** password. If you connect to the Daemon over the network, you have root (sudo) rights.

Further Information here.

Expose the Daemon to TCP

1. Navigate to `/etc/docker/`.
2. Create new File `daemon.json` in the directory.

```
{
 "hosts": [
"unix:///var/run/docker.sock","tcp://0.0.0.0:2376
"],
 "log-driver": "journald"
 }
```

3. Restart the Daemon: `sudo systemctl stop docker` -> `sudo dockerd`
4. Check if you are able to access the daemon from your Computer by opening a terminal and typing `curl <VM-IP>:2376/version`. You should now see som Information about the daemon.

Generate Keys and Certificates

1. First, generate an CA RSA private key `openssl genrsa -aes256 -out ca-key.pem 4096`
2. Enter a pass phrase for your private key, e.g. `MyPassword`
3. Generate a certificate from the private key: `openssl req -new -x509 -days 365 -key ca-key.pem -sha256 -out ca.pem`
4. Enter a pass phrase. Then you can leaf blank the country code and the following except **common name**. There you enter the *VM's IP e. g. 192.168.1.100*.
5. Once you have generated the CA keys, you can generate the server keys. Make sure that you **always use the same common name** as above.
6. Generate the server key: `openssl genrsa -out server-key.pem 4096`
7. Create a server key and certificate signing request (CSR) and replace the *common-name* with your common name: `openssl req -subj "/CN=<common-name>" -sha256 -new -key server-key.pem -out server.csr`
8. Sign the public key wit the CA. Replace the *notebook-IP* with the IP of your Host Machine and the *common-name* with your common name. `echo subjectAltName = DNS:<common-name>,IP:<notebook-IP>,IP:127.0.0.1 >> extfile.cnf`
9. Set the Docker daemon key's extended usage attributes to be used only for server authentication: `echo extendedKeyUsage = serverAuth >> extfile.cnf`
10. Create the signed certificate:

```
openssl x509 -req -days 365 -sha256 -in
server.csr -CA ca.pem -CAkey ca-key.pem -
CAcreateserial -out server-cert.pem -extfile
extfile.cnf
```

    and enter a pass phrase.

11. Create a new key for the Client: `openssl genrsa -out key.pem 4096`
12. Generate a client CSR: `openssl req -subj '/CN=client' -new -key key.pem -out client.csr`
13. To make the key suitable for client authentication, create a new extensions config file: `echo extendedKeyUsage = clientAuth > extfile-client.cnf`
14. Generate certificate for the client signed by the CA:

```
openssl x509 -req -days 365 -sha256 -in
client.csr -CA ca.pem -CAkey ca-key.pem   -
CAcreateserial -out cert.pem -extfile extfile-
client.cnf
```

    and enter a pass phrase

15. For a clearer overview, you can delete some files when you created *cert.pem* and the *server-cert.pem*: `rm -v client.csr server.csr extfile.cnf extfile-client.cnf`

Daemon TLS Security

1. Stop Docker using `sudo systemctl stop docker`
2. Restart the Daemon from the directory containing the keys and certificates. `dockerd --tlsverify --tlscacert=ca.pem --tlscert=server-cert.pem --tlskey=server-key.pem`
3. Now try to connect from your host machine to the Daemon: `curl <VM-IP>:2376/version`. You should receive an error that you are trying to connect to a HTTPS server over HTTP.
4. Then try `curl https://<VM-IP>:2376/version`. You should nor receive an error about a missing certificate.
5. You now successfully protected your Daemon.

Connect to Daemon using Certificate

1. If you are running Windows (otherwise continue with step 2) on your host machine you are not able to `curl` with SSL certificates. Instead you can run the following command on the VM.

```
curl -k https://127.0.0.1:2376/version --cert
cert.pem --key key.pem --cacert ca.pem
```

2. First, you need to export the client credentials. Copy the **ca.pem**, **cert.pem** and **key.pem** to your host machine. You might use simplefileexchange.com.
3. On your host machine open a console in the directory containing the keys. Connect to the Daemon with the certificate.

```
curl -k https://<VM-IP>:2376/version --cert
cert.pem --key key.pem --cacert ca.pem
```

Task 4 - Container Outbreak

Table of Content

Introduction

If a Docker Container runs in with the `root` user, it has unrestricted access to all resources of the host's OS.

Read the Sudoers File

1. Watch the Volumes in the Attackers *docker-compose.yml*. Here you can see that the Host's `/etc` Folder was mount to the Docker Container.
2. Start the **Attacker App** if it is not running.
3. Navigate to the virtual Terminal in the Attacker App: **Container Outbreak** -> **Terminal**
4. Type `whoami` to see what user is running the container. User should be: `root`
5. Now press the **show Sudoers File** to display the content of the *sudoers* file. Alternatively, you can enter `cat ../../etc/sudoers` in the Terminal.

Change the User

1. Configure a new user in the Dockerfile from the Attacker App. Add all the lines with a comment above.

```
FROM node:12

#Add new Group "secure", or do nothing if already
exists
RUN groupadd -r secure || true
#Add new User "secure", or do nothing if already
exists
RUN useradd -r -s /bin/false -g secure secure ||
true

WORKDIR /var/attackerApp

COPY ./AttackerApp /var/attackerApp/

COPY ./README.md /var/
```

```
COPY ./Task01_PublicAccess
/var/Task01_PublicAccess
COPY ./Task02_InterContainerComm
/var/Task02_InterContainerComm
COPY ./Task03_TLSforDaemon
/var/Task03_TLSforDaemon
COPY ./Task04_ContainerOutbreak
/var/Task04_ContainerOutbreak
COPY ./Task05_DoS /var/Task05_DoS
COPY ./Task06_DockerSecrets
/var/Task06_DockerSecrets
COPY ./Task07_ImageTag /var/Task07_ImageTag

RUN npm install

#Change owner of the working directory to the
"secure" user
RUN chown -R secure:secure /var
#Proceed with user "secure"
USER secure

EXPOSE 2222
ENTRYPOINT [ "npm", "start" ]
```

2. Type `whoami` to see what user is running the container. User should now be: `secure`
3. Try again to open the sudoers file
4. You should receive a *Permission denied*.

Task 5 - Denial of Service

Table of Content

Introduction

By default, Docker Containers share all the host OS's resources with no limitations. To limit CPU usage or memory, you have to specify the maximum usage for example in the docker-compose.yml. It is also possible to grant some usage to a Container.

More information can be found on the Docker Compose reference.

Execute Denial of Service

1. Install a System Monitor: `sudo apt-get install gnome-system-monitor`
2. Start the System Monitor (you can search by pressing the "Start" Button). You can see the CPU usage under the "Resources" tab.
3. Navigate to *05 Denial of Service* of the Attacker App and press the *Block CPU* button.
4. Observe the CPU usage of the VM. It should raise to **100%** for about 15 seconds.

Limit the Max CPU Usage

You can limit different resources in the `docker-compose.yml`.

1. Stop the Container containing the Attacker App
2. Open the *docker-compose.yml* from the Attacker App: `nano AttackerApp/docker-compose.yml`
3. Add the *deploy* section to the Container. Here you can limit the amount of CPUs.

```
deploy:
   resources:
     limits:
       cpus: "0.50"
```

4. Restart the Container and execute the DoS attack again. Observe the CPU usage now. It should be limited to 50% for the Attacker App.

Task 6 - Docker Secrets

Table of Content

Introduction

For many applications it is necessary to be able to perform a password authentication, for example for database access. This password should never be programmed directly into the source code, as it could be cracked by reverse engineering. Docker offers a special Secrets Management for this purpose.

Add Password Protection to mongoDB

If you have not applied the password protection for the DB in task 2, you have to do it now.

1. Add the user/password environment variables to the *docker-compose.yml* of *ExpressCart*.

```
environment:
  - MONGO_INITDB_ROOT_USERNAME=user
  - MONGO_INITDB_ROOT_PASSWORD=password
  - MONGO_INITDB_DATABASE=expresscart
```

2. Change the **databaseConnectionString** in *expressCart/config/settings.json* to `mongodb://<username>:<password>@mongodb:27017/expresscart?authSource=admin`.
3. Unfortunately it is not possible to add a user to an existing database in mongodb, only using Docker Compose. Therefore, the data needs to be deleted and recreated.

```
sudo docker container prune #delete all stopped
Containers
sudo docker volume rm expresscart_expresscart-
mongo-data #delete the Volume
```

4. Afterwards, the Application can be restarted. `sudo docker-compose up --build`
5. Add the Sample Data

```
sudo docker exec -it expresscart bash
npm run testdata
exit
```

6. The DB Connection is now Authenticated.

Store Secrets in an external File

1. Create a new file in the *expressCart* for the DB password called *mongodb* with just the password in it.

```
password
```

2. Add the *mongodb*-password as a secret to the *docker-compose.yml*. Just change the following lines:

```
version: '3'
mongodb:
 environment:
   -
MONGO_INITDB_ROOT_PASSWORD_FILE=/run/secrets/mong
o_pass
 secrets:
   - mongo_pass

secrets:
  mongo_pass:
    file: ./mongodb
```

3. Again, delete the database and recreate it.:

```
sudo docker container prune #delete all stopped
Containers
sudo docker volume rm expresscart_expresscart-
mongo-data #delete the Volume
```

4. Restart the Application: `sudo docker-compose up --build`
5. Add the Sample Data

```
sudo docker exec -it expresscart bash
npm run testdata
exit
```

Important

Add the *mongodb* file to the .ignore*. Otherwise, it makes no sense to protect/outline the passwords.

Task 7 - Image Tag

Table of Content

Introduction

The Docker Hub provides various images that can be used as the basis for your own images. The images identify themselves with a name and a tag indicating the "version". It is possible to add the tag name or the hash of the image. You also always have the possibility to leave the Tag, which automatically uses the *latest* tag.

Browse Docker Hub

Go on Docker Hub and watch the node images. Open the Tag section and notice the date of the last change. Now search for the *latest* tag and compare the date of the last change. The node developers normally update the *latest* tag.

On many other images, the image with the *latest* tag is not the latest pushed image. Search some other publishers to find a repositories with *latest* tagged images that are outdated.

Therefore, it is always recommended to specify the image at least with a specific tag.

Tag Image with Hash

1. Go to Docker Hub and search for the **node:12** image which is used in the Attacker App.
2. Copy its SHA256 Hash to your clipboard.
3. Change the tag of the base image in the AttackerApp's Dockerfile:

```
#replace
FROM node:12
#with
FROM
node@sha256:beb72fb7dab30898dad23419a58e1f995a636
4b4c8b000100715aa190c323392
```

4. Rebuild the AttackerApp to prove it is working.