

# Sicherheitsaspekte der Plattform ScaleIT im Kontext von DigitalTWIN



# Sicherheitsaspekte der Plattform ScaleIT im Kontext von DigitalTWIN

Hannah Wester, Michael Heintl, Peter Schneider

4. August 2020

Erstellt im Auftrag von

**se** | commerce



Version 1.0

Fraunhofer-Institut für Angewandte und Integrierte Sicherheit  
[www.aisec.fraunhofer.de](http://www.aisec.fraunhofer.de)

Fraunhofer AISEC  
Lichtenbergstraße 11  
85748 Garching (bei München)

Kontakt Autoren  
[hannah.wester@aisec.fraunhofer.de](mailto:hannah.wester@aisec.fraunhofer.de)  
[michael.heinl@aisec.fraunhofer.de](mailto:michael.heinl@aisec.fraunhofer.de)

Kontakt Abteilungsleitung Produktschutz & Industrial Security  
[bartol.filipovic@aisec.fraunhofer.de](mailto:bartol.filipovic@aisec.fraunhofer.de)

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>4</b>
<b>2</b>	<b>Bestandteile der ScaleIT Plattform</b>	<b>6</b>
2.1	Offene Ports . . . . .	6
2.2	Docker . . . . .	6
2.3	Docker Compose . . . . .	7
2.4	Rancher . . . . .	7
2.5	ScaleIT App-Registry (Verzeichnisdienst ETCD) . . . . .	8
2.6	Go Git Service . . . . .	8
2.7	Dokumentation . . . . .	8
<b>3</b>	<b>Kommunikation und Erreichbarkeit</b>	<b>9</b>
3.1	Zugang zum Host . . . . .	9
3.2	Zugang von außen . . . . .	11
3.2.1	Absicherung von HTTP . . . . .	11
3.2.2	Absicherung von MQTT . . . . .	12
3.3	Kommunikation zwischen Apps einer ScaleIT Instanz . . . . .	13
3.3.1	Isolation von Apps . . . . .	13
3.3.2	Zulassen legitimer Kommunikation . . . . .	15
3.3.3	Netzwerkzugriffsmöglichkeit durch Apps . . . . .	17
<b>4</b>	<b>Leitfaden für Entwickler</b>	<b>19</b>
4.1	Sicherer Entwicklungsprozess . . . . .	19
4.2	Transparenz . . . . .	20
4.3	Secure Coding . . . . .	21
4.4	Security Testing . . . . .	22
4.5	Sichere Docker-Entwicklung . . . . .	23
4.6	Datenverschlüsselung . . . . .	24
4.7	Produktschutz . . . . .	24
<b>5</b>	<b>Leitfaden für Ökosystembetreiber</b>	<b>26</b>
5.1	Automatische (Sicherheits-)Updates . . . . .	26
5.2	Vollverschlüsselung von Datenträgern . . . . .	27
5.3	Allgemeine Reduzierung der Angriffsfläche . . . . .	29
5.4	Transport Layer Security . . . . .	29
5.5	Secure Shell . . . . .	30
5.6	Antivirensoftware . . . . .	32
5.7	Docker Konfiguration . . . . .	33
5.8	Backups . . . . .	33
5.9	Security Testing . . . . .	34
<b>6</b>	<b>Fazit</b>	<b>36</b>

# 1 Einleitung

Das Projekt *digitalTWIN* hat sich zum Ziel gesetzt, auf die Anforderungen des Bauwesens zugeschnittene, IT-gestützte Werkzeuge zu entwickeln. Hierbei findet unter anderem die aus einem vom Bundesministerium für Bildung und Forschung (BMBF) geförderten Projekt hervorgegangene Plattform *ScaleIT* Verwendung.

Es existieren mehrere Implementierungsvarianten von *ScaleIT*, die sich unter anderem durch den Einsatz verschiedener Kombinationen von Plattformen zur Orchestrierung und zum Containermanagement wie zum Beispiel Rancher, Cattle und Kubernetes voneinander unterscheiden.<sup>1</sup> Der vorliegende Leitfaden befasst sich mit verschiedenen sicherheitsrelevanten Aspekten der in Form einer virtuellen Maschine (VM) für die Virtualisierungsplattform *VirtualBox* frei erhältlichen *ScaleIT Community Edition*<sup>2</sup> (CE). Zwar können die Erkenntnisse dieses Leitfadens zum Teil auch auf die nicht frei erhältliche *ScaleIT Enterprise Edition* (EE) und ggf. andere Varianten von *ScaleIT* übertragen werden, allerdings standen diese im Rahmen dieser Arbeit nicht zur Verfügung.

Neben dem „klassischen“ Fall einer direkten Infiltration durch außenstehende Dritte wird, wie in Abbildung 1.1 dargestellt, unter anderem auch auf alternative Angreifermodelle, wie z. B. nicht vertrauenswürdige App-Entwickler oder indirekte Infiltration mittels *Lateral Movement* über kompromittierte Systeme des Ökosystembetreibers eingegangen (sogenannte *Supply-Chain-Angriffe*). Zudem ist zu beachten, dass *digitalTWIN* den Einsatz der *ScaleIT*-Plattform auf der Baustelle vorsieht. Daraus ergibt sich sowohl die Möglichkeit eines physischen Zugriffs als auch die (gewünschte) Nutzung durch verschiedene Stakeholder mit unterschiedlichen Berechtigungen, die zwar nicht auf alle Services und Daten gleichermaßen Zugriff haben sollen, jedoch mobile, möglicherweise mit Schadsoftware infizierte Drittgeräte nutzen.

Aufgrund der Vielfältigkeit der eingesetzten Komponenten sowie deren komplexem Zusammenspiel ist eine abschließende Bewertung der Sicherheit des Gesamtsystems im limitierten Rahmen dieses Leitfadens nicht möglich. Vielmehr ist die Aufgabe des vorliegenden Dokuments, ein Grundverständnis für die Funktionen der verwendeten Komponenten sowie deren Interaktion untereinander zu schaffen und darauf aufbauend sowohl Entwickler als auch Ökosystembetreiber für potentielle Risiken zu sensibilisieren und ihnen Handlungsempfehlungen zu geben.

---

<sup>1</sup>[https://scale-it.org/de/downloads-697.html?file=files/cto\\_layout/img/Downloads/ScaleIT\\_gemeinsamer%20Abschlussbericht.pdf](https://scale-it.org/de/downloads-697.html?file=files/cto_layout/img/Downloads/ScaleIT_gemeinsamer%20Abschlussbericht.pdf)

<sup>2</sup><https://github.com/scaleit-i40/scaleit-ce-vm>

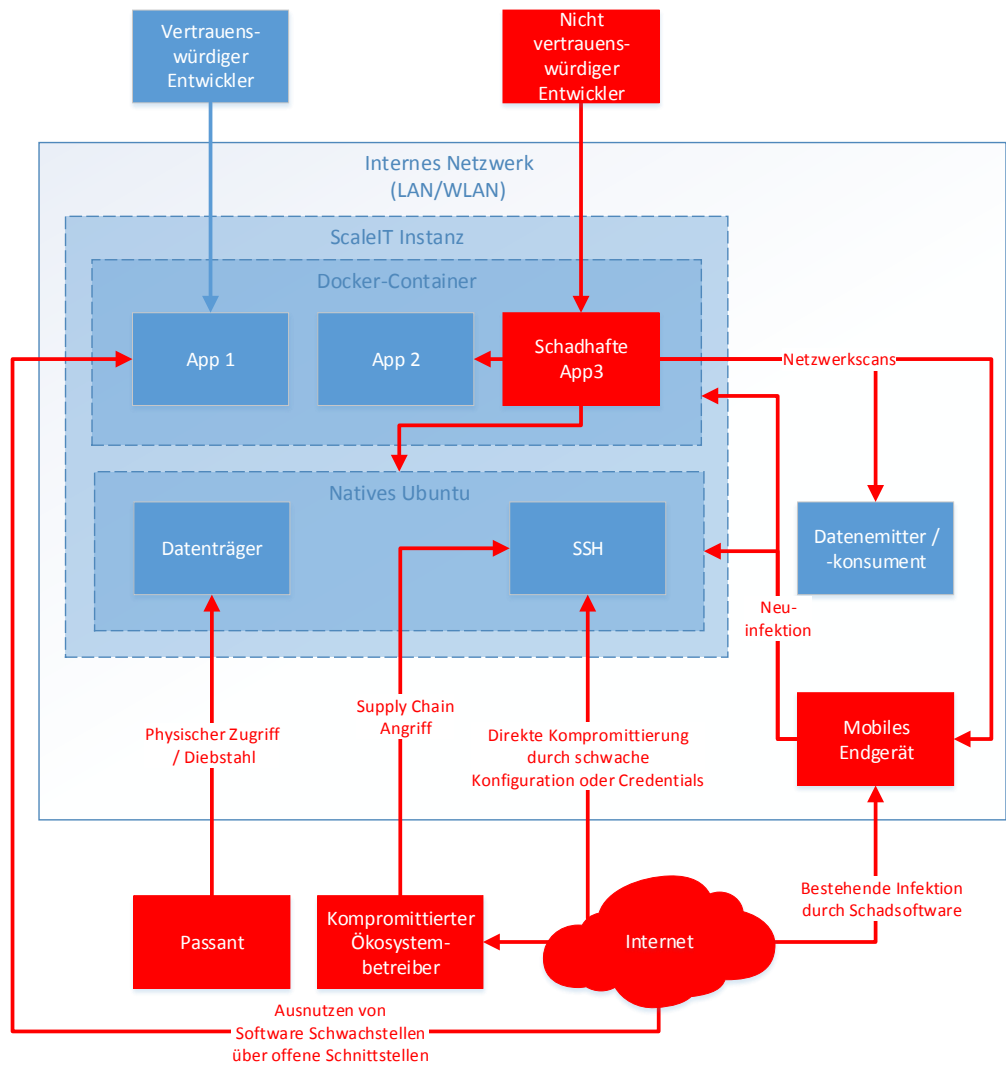


Abbildung 1.1: Beispielhafte Angriffsszenarios im DigitalTWIN-Kontext.

## 2 Bestandteile der ScaleIT Plattform

In diesem Abschnitt werden die wesentlichen technischen Bestandteile von ScaleIT kurz beleuchtet.

### 2.1 Offene Ports

Um eine erste Übersicht der in ScaleIT laufenden Dienste zu erhalten, wird mithilfe des Werkzeugs *Nmap*<sup>3</sup> vom Host aus ein Portscan der laufenden virtuellen Maschine durchgeführt. Die in Tabelle 2.1 dargestellten Ergebnisse zeigen, dass insgesamt 13 Ports offen sind. Grundsätzlich sind offene Ports für die Kommunikation notwendig und können im Regelfall deshalb nicht gänzlich verhindert werden. Allerdings ist es wichtig, die Dienste, die über die jeweiligen Ports erreichbar sind, entsprechend gegen unberechtigte Zugriffe zu schützen.

Tabelle 2.1: Durch Nmap-Portscan identifizierte offene Ports der ScaleIT CE VM.

<b>Port</b>	<b>Dienst</b>	<b>Initiiert durch</b>
22	SSH	nativ
80	ScaleIT Launchpad	Rancher
1883	MQTT	Rancher
1884	MQTT	Rancher
3000	Gogs	Docker
8080	Rancher	Docker
10022	Gogs SSH	Docker
49502	ETCD Web-Service	Rancher
49501	ETCD Service	Rancher
51530	Node-RED	Rancher
51531	Node-RED	Rancher
51515	ScaleIT Launchpad	Rancher
51556	DTA Simulation Temperatur-Messung	Rancher

### 2.2 Docker

Durch einen gemeinsamen Kernel und einen Dienst als Middleware bietet Docker eine Basis für die Isolation verschiedener Anwendungen. Benötigte Bibliotheken,

---

<sup>3</sup><https://nmap.org/>

Laufzeitumgebungen und Ressourcen können von der Middleware jeder Anwendung einzeln zugeordnet werden. Dies erlaubt den parallelen Betrieb von Anwendungen mit unterschiedlichsten und auch konkurrierenden Anforderungen ohne den Einsatz komplexer Virtualisierungstechniken. Eine Anwendung läuft dabei zusammen mit ihren Ressourcen in einem Container. Auf technischer Ebene ist die erzielte Isolation jedoch geringer als sie durch virtuelle Maschinen erreichbar ist. Durch die gemeinsame Nutzung des Kernels und von Systemressourcen kann es zu Konflikten und auch Möglichkeiten zur Manipulation kommen. Sicherheitsprobleme sind zuerst einmal auf einzelne Container und ihre Daten beschränkt. Durch unsichere oder unsaubere Konfiguration sowie durch Fehler in der Middleware oder aber auch dem Kernel des Host-Betriebssystems kann die Isolation der Container verloren gehen und weitere Manipulationen nach sich ziehen. Eine Möglichkeit Konfigurationsfehler zu vermeiden, stellt die Erstellung dedizierter Sicherheitsprofile für jeden Container dar. Hierbei werden für jeden Container erlaubte Aktionen in einer Whitelist aufgeführt. Abschnitt 5.7 gibt Hinweise, um Docker sicher einzurichten.

## 2.3 Docker Compose

Docker Compose ist ein Tool, um Multi-Container-Anwendungen zu definieren und auszuführen. In der Datei `docker-compose.yml` kann unter anderem festgelegt werden, welche Services gemeinsam gestartet werden sollen, wie diese Services verknüpft sind, welche Netzwerke für die Kommunikation zwischen den Containern verwendet werden und wie die Datenablage organisiert ist.

## 2.4 Rancher

Rancher ist eine grafische Benutzeroberfläche zur Verwaltung von Multi-Container-Anwendungen, die auch auf unterschiedlichen Hosts laufen können. Rancher selbst wird ebenfalls als Docker-Container ausgeführt. Rancher bietet in einem App-Store Docker-Anwendungen an, die einfach über das User Interface (UI) integriert werden können. Anstelle des öffentlichen App-Stores wurden bei ScaleIT ein App-Store des Anbieters sowie ein lokaler App-Store integriert, der über das Git-Repository Gogs mit eigenen Anwendungen befüllt werden kann. Üblicherweise werden die im ScaleIT-Kontext als *Apps* bezeichneten Multi-Container-Anwendungen in Rancher als *Stacks* abgebildet. Ein *Stack* besteht dabei aus mehreren *Services*, welche jeweils wiederum aus einem oder mehreren (Docker-) *Containern* bestehen.

Rancher 2.x setzt auf dem Orchestrierungsframework Kubernetes auf. In der von ScaleIT verwendeten Version 1.6.27 setzt Rancher standardmäßig auf Cattle<sup>4</sup> auf, das kompatibel zur Docker Compose Standard-Syntax ist. Rancher 1.x

<sup>4</sup><https://github.com/rancher/cattle>



wird allerdings seit Anfang 2020 nicht mehr gewartet. Für die Verwendung von Rancher mit Cattle wird die Docker Compose Datei in *docker-compose.yml.tpl* umbenannt und um ein *rancher-compose.yml* erweitert, worin weitere, rancher-spezifische Informationen z. B. zum Netzwerk angegeben werden können.

### 2.5 ScaleIT App-Registry (Verzeichnisdienst ETCD)

Die ScaleIT App-Registry basiert auf dem Verzeichnisdienst *etc distributed* (ETCD<sup>5</sup>). Die App-Registry ist nicht mit der Docker-Registry zu verwechseln, aus der Docker Images bezogen werden können und für die es in Rancher ebenfalls die Möglichkeit gibt, Zugangsdaten zu hinterlegen. Die App-Registry hingegen wird genutzt, um über das Launchpad laufende Services einem Nutzer ohne Zugriffsrechte auf Rancher zur Verfügung zu stellen.

Durch Nutzung des von ScaleIT bereitgestellten *Registration Sidecars*<sup>6</sup>, welches in die Datei *docker-compose.yml* einer jeden App aufgenommen werden sollte, können sich Apps während des Starts in der App-Registry registrieren. In der Docker Compose Datei können über Key / Value-Paare Optionen für den Verzeichnisdienst konfiguriert werden.

### 2.6 Go Git Service

*Go Git Service* (Gogs) ist ein selbst gehosteter Git Dienst, der auf Port 3000 in ScaleIT läuft. Das Gogs-Repository *scaleit/rancher-templates* fungiert dabei als lokaler App-Pool für Rancher, anhand dessen selbst entwickelte oder modifizierte Apps lokal hochgeladen und im Rancher-Katalog zur Verfügung gestellt werden können. Auf Port 10022 läuft hierfür ein spezieller Secure Shell (SSH)-Dienst, der es Entwicklern zusätzlich erlaubt, mit dem Repository zu interagieren, ohne dafür Zugriff auf den nativen SSH-Dienst auf Port 22 bekommen zu müssen.

### 2.7 Dokumentation

ScaleITs Dokumentation und Repository befinden sich auf <https://github.com/ScaleIT-ORG/>. Als weitere Informationsquellen werden ein Wiki<sup>7</sup> sowie ein Repository<sup>8</sup> zur Verfügung gestellt, aus dem man u. A. ScaleIT CE als VM herunterladen kann. Während der Recherchen für diesen Leitfaden wurde hauptsächlich mit dieser VM gearbeitet.

---

<sup>5</sup><https://etcd.io/>

<sup>6</sup><https://github.com/scaleit-i40/registration.git>

<sup>7</sup><https://wiki.scaleit-i40.de>

<sup>8</sup><https://github.com/scaleit-i40>

## 3 Kommunikation und Erreichbarkeit

Wie bereits beschrieben, besteht die ScaleIT Architektur aus einer Reihe von Microservices, die miteinander kommunizieren. Je nachdem, wie sie angeschlossen sind, können diese aus dem internen Firmennetz oder direkt aus dem Internet über den *Nginx*<sup>9</sup> Reverse Proxy erreicht werden. In ScaleIT CE ist der Reverse Proxy über Hypertext Transfer Protocol (HTTP) erreichbar und leitet auf die entsprechenden Ports der Services weiter. Es sind keine Security-Mechanismen auf dem Nginx implementiert. In ScaleIT EE kommuniziert Nginx mit einem Domain Name System (DNS)-Service, der auf dem Host läuft, um die IP-Adressen der Services auf Domain-Namen statt auf Ports zu mappen. Zusätzlich verwaltet er Zertifikate, um die Kommunikation von außen mit TLS abzusichern. Die Infrastruktur-Services laufen, wenn notwendig, in einem von Docker bereitgestellten Bridge- oder Host-Netzwerk. Die User-Services laufen in einem separaten Netzwerk, das von Rancher über das Container Network Interface (CNI) verwaltet wird (siehe dazu auch Abschnitt 3.3). Von außen können sie über ihre Ports bzw. über ihren Domain-Namen aufgerufen werden. Die beschriebene Netzwerkarchitektur ist in Abbildung 3.1 veranschaulicht.

Ein Microservice kann grundsätzlich über den Host, offene Ports nach außen und innerhalb des Docker-Netzwerkes erreicht werden. Deshalb ist es einerseits wichtig, den Host abzusichern und andererseits nur notwendige Ports zu öffnen sowie diese ebenfalls entsprechend abzusichern. Dies gilt nicht nur für den eigenen Service, sondern für alle Services, die im gleichen Netzwerk gestartet werden. Für den Fall, dass die Vertrauenswürdigkeit aller Services nicht sichergestellt werden kann, muss eine Isolierung von Services durch Segmentierung des Netzwerks und durch eine Authentifizierung der Kommunikationspartner erfolgen.

### 3.1 Zugang zum Host

Der Host kann physisch oder über offene Dienste von außen erreicht werden. Beide Zugangsmöglichkeiten müssen auf autorisierte Nutzer beschränkt werden. Die Absicherung des Hardware-Zugriffs ist insbesondere wichtig, da ScaleIT im digitalTWIN-Kontext auf der Baustelle verwendet wird und somit physisch nur begrenzt isoliert werden kann. Somit muss die Anmeldung am Host abgesichert sein. Zusätzlich sind eine Festplattenverschlüsselung und ein Backup sinnvoll, um Datenextraktion und -löschung vorzubeugen.

---

<sup>9</sup><https://www.nginx.com/>

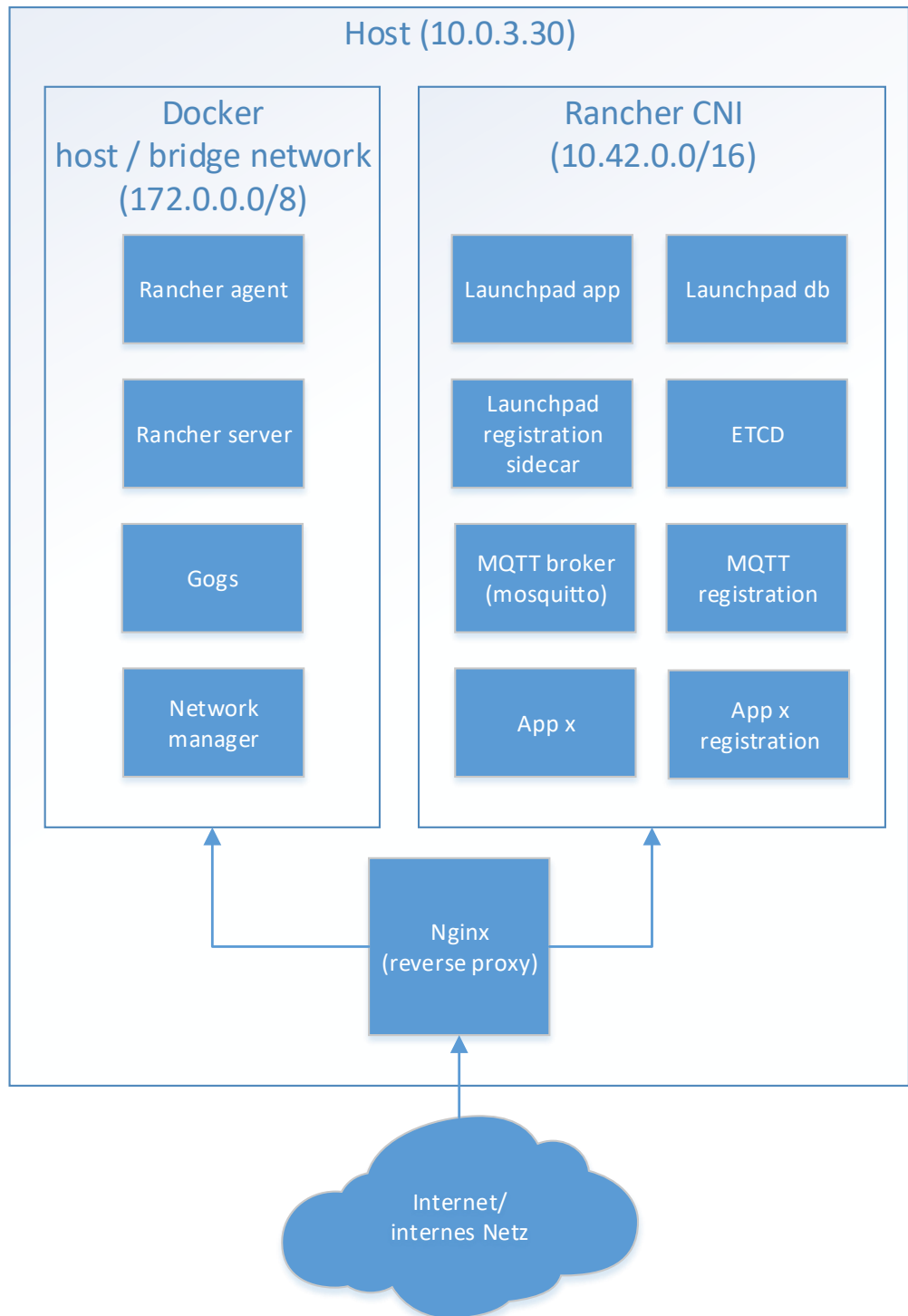


Abbildung 3.1: ScaleIT Netzwerkarchitektur

Wie während des Portscans festgestellt, läuft auf dem Host ein SSH Dienst (Tabelle 2.1). Für diesen ist der root-Login auf asymmetrische Schlüsselpaare beschränkt.

In der ScaleIT CE VM kann mit kurzzeitigem Zugriff auf den Host das root-Passwort überschrieben werden. Mit einem Boot in den Einzelbenutzermodus gelangt man bei der Standardinstallation von Ubuntu direkt in eine Notfallkonsole, die über root-Rechte verfügt und in der Lage ist, das Passwort neu zu setzen. Danach lässt sich entsprechend auch die Konfiguration des SSH-Dienstes dahingehend manipulieren, dass ein Remotelogin mit root-Rechten möglich ist. Der root-Zugang muss zu Beginn der Nutzung von ScaleIT abgesichert werden. In Kapitel 5 wird beispielhaft beschrieben, wie der Host für diesen Zweck konfiguriert werden könnte.

## 3.2 Zugang von außen

Standardmäßig wird ein Service von einem Nutzer im Browser aufgerufen und muss somit über HTTP erreichbar sein. Zusätzlich sollen Sensoren und IoT-Geräte an die ScaleIT-Plattform angeschlossen werden, was durch ein serviceorientiertes Publish / Subscribe-Protokoll wie Message Queuing Telemetry Transport (MQTT) erleichtert wird. Unter Umständen muss eine weitere ScaleIT Instanz oder weitere Datenemitter / -konsumenten über HTTP (REST) oder Websocket angeschlossen werden. Eingehender Netzwerkverkehr wird durch einen Reverse Proxy zu den Services weitergeleitet.

Alle offenen Schnittstellen, über die vertrauliche Daten geschickt werden, sollen verschlüsselt sein und der Empfänger soll sich authentifizieren. Alle offenen Schnittstellen, über die Daten gesendet werden, die nicht verändert werden dürfen (z. B. weil ansonsten durch fehlerhaftes Monitoring Schäden entstehen können), sollen vom Sender signiert werden. Wenn Services zeitkritische Dienste ausführen, soll dafür gesorgt werden, dass sie nicht mit Nachrichten geflutet werden können - z. B. indem Daten von nicht autorisierten Sendern vom Proxy geblockt werden oder durch eine niedrigere Priorisierung der Daten von außen.

### 3.2.1 Absicherung von HTTP

Über HTTP gelangt man zum Reverse Proxy, der die Anfrage zum entsprechenden Service weiterleitet. In ScaleIT EE ist Nginx bereits mit einem selbstsignierten Zertifikat vorkonfiguriert, sodass die Kommunikation zu den Services ebenfalls über HTTPS erfolgen kann. Dieses Zertifikat sollte durch ein eigenes (von der Certificate Authority (CA) / Public Key Infrastructure (PKI) des Ökosystembetreibers signiertes) Zertifikat ersetzt werden, sodass eine korrekte Verifizierung des Zertifikats möglich ist. Zudem ist es wichtig, dass TLS richtig konfiguriert ist, siehe dafür Abschnitt 5.4.

Weiterhin sollte eine Client-Authentifizierung stattfinden, sodass nur bestimmte Nutzer auf bestimmte Apps zugreifen können. Dies ist wichtig, da das System von unterschiedlichen Stakeholdern mit unterschiedlichen Zugriffsrechten genutzt werden soll. Um dies nicht für jeden Service einzeln umsetzen zu müssen, kann ein SSO-Verfahren über einen Identity Provider (IDP) an einer zentralen Stelle konfiguriert werden. In dem IDP kann ein Rechte- und Rollenkonzept umgesetzt werden, sodass ein Nutzer beim IDP registriert und einer bestimmten Gruppe mit bestimmten Rechten zugewiesen werden kann. Wichtig ist, dass die Authentifizierung des Nutzers gegenüber dem IDP sicher geschieht. Dies kann z. B. mit einem Passwort unter Erzwingung einer ausreichenden Mindestkomplexität und -länge sowie einer beschränkten Anzahl an Login-Versuchen erfolgen. Noch sicherer ist jedoch, die Verwendung von Multi-Faktor-Authentisierung (MFA), wie z. B. einer Smartcard mit digitalem Zertifikat, welche zusätzlich mit einer persönlichen Identifikationsnummer (PIN) abgesichert ist. Prominente Beispiele für kostenfreie IDPs sind Shibboleth und Keycloak, die Protokolle wie SAML, oAuth oder OpenID Connect unterstützen. In ScaleIT EE ist die Client-Authentifizierung über Nginx an einer zentralen Stelle umgesetzt, sodass es die Möglichkeit gibt, bestimmte Apps mit einer Zugriffskontrolle zu schützen. Der Proxy leitet einen Nutzer, der auf eine App zugreifen möchte, zum Authorisierungs-Server Keycloak weiter. Dort muss sich ein Nutzer im Browser (einmalig) authentisieren und wird anschließend auf die Seite der Webapplikation weitergeleitet, falls er die entsprechenden Berechtigungen besitzt.

#### 3.2.2 Absicherung von MQTT

In ScaleIT ist der MQTT-Broker *mosquitto*<sup>10</sup> bereits als Service vorinstalliert und über die Ports 1883 / 1884 erreichbar. Somit kann zusätzlich zu HTTP auch über MQTT kommuniziert werden, sobald ein Service einen MQTT-Client implementiert hat. Der vorinstallierte Service Grafana ermöglicht die visuelle Darstellung von eingehenden Daten. Die API dafür kann mittels NodeRED modifiziert werden. Die Kommunikation über MQTT ist zunächst weder authentisiert noch verschlüsselt. Wenn die Daten, die eingebracht werden, vertraulich sind oder man sich auf die Daten verlassen muss, müssen zusätzliche Mechanismen angewandt werden.

Dies kann ähnlich wie für HTTP umgesetzt werden: Ein Zertifikat für TLS kann entweder für den Proxy oder für den Broker konfiguriert werden. Geräte, die von außerhalb Daten mit Anforderungen hinsichtlich Integrität und Authentizität einbringen, sollten sich entsprechend anhand eines Client-Zertifikats authentifizieren. Vertrauliche Topics dürfen nur von autorisierten Nutzern abonniert werden, dafür kann über einen IDP ebenfalls ein Rollen- und Rechtekonzept umgesetzt werden. Im Broker können Berechtigungsmechanismen aktiviert werden, z. B. mit welchem Quality of Service Level ein Client senden / abonnieren darf und welche Topics von welchem Client abonniert / gesendet werden dürfen. Dies sollte

---

<sup>10</sup><https://mosquitto.org/>

mit einer Whitelist umgesetzt werden, sodass ohne explizite Freischaltung keine Kommunikation zu bestimmten Topics erlaubt ist. Ein Client kann die maximale Nachrichtengröße außerdem beschränken, um Überlassungsangriffen vorzubeugen.

### 3.3 Kommunikation zwischen Apps einer ScaleIT Instanz

Innerhalb der Docker-Welt gibt es verschiedene Konfigurationsmöglichkeiten für die Kommunikation zwischen Anwendungen in unterschiedlichen Containern. Während Docker standardmäßig - und wenn nicht anders definiert - jeden Container derselben `default bridge` hinzufügt<sup>11</sup>, erstellt `docker-compose` für jede in einer `docker-compose.yml`-Datei definierte App ein separates `bridge`-Netzwerk, über welches die in dieser App enthaltenen Container/Services standardmäßig miteinander kommunizieren können.

Das in ScaleIT verwendete *Rancher* in der Version 1.6.27 verwendet zur Verwaltung von Apps, die aus einem oder mehreren Containern bestehen können, zwar `docker-compose`, die Konfiguration von Netzwerkparametern erfolgt allerdings anhand des CNI-Frameworks. Docker-Container, welche durch *Rancher* gestartet werden, werden deshalb mit dem `-net=none`-Parameter ausgeführt, um Dockers Netzwerk-Mechanismen zu umgehen. Dies macht sich operativ unter anderem dadurch bemerkbar, dass über *Rancher* freigegebene Ports nicht über die üblichen Kommandozeilenwerkzeuge wie `docker ps` oder `netstat` ersichtlich sind. Stattdessen werden die Container vom CNI standardmäßig per Overlay-Netzwerk unter Verwendung von IPsec-Tunneling verbunden.<sup>12</sup>

Im folgenden wird die Absicherung der Kommunikation zwischen Apps durch Cattle beschrieben. Wenn ein anderes Orchestrierungs-Tool verwendet wird, muss die Absicherung entsprechend analysiert werden. Allgemein lässt sich sagen, dass Docker Netzwerke und CNIs von außen nicht erreichbar sind. Wenn man davon ausgeht, dass alle Microservices vertrauenswürdig und so gehärtet sind, dass sie nicht von außen kompromittiert werden können, ist eine Absicherung der Kommunikation innerhalb des Netzwerkes nicht unbedingt nötig. Wie in Abbildung 1.1 gezeigt, gibt es in unserem Angreifermodell allerdings auch eine schadhafte App, weshalb die Kommunikation innerhalb ähnlich der Kommunikation von/nach außen abzusichern ist.

#### 3.3.1 Isolation von Apps

Um eine etwaige Isolation der Apps untereinander zu testen, wird eine prototypische App entwickelt, die auf dem bekannten Portscanner *Nmap-Portscan*<sup>13</sup> bzw.

<sup>11</sup><https://docs.docker.com/network/bridge/>

<sup>12</sup><https://rancher.com/docs/rancher/v1.6/en/rancher-services/networking/>

<sup>13</sup><https://nmap.org/>

The screenshot shows the output of an Nmap scan. Key findings are highlighted with red boxes:

- 4/17/2020 9:11:36 AM Scanning 10.42.20.214 [1000 ports]
- 4/17/2020 9:11:36 AM Discovered open port 3306/tcp on 10.42.20.214
- 4/17/2020 9:11:43 AM Not shown: 999 closed ports
- 4/17/2020 9:11:43 AM PORT STATE SERVICE VERSION
- 4/17/2020 9:11:43 AM 3306/tcp open mysql MySQL 5.7.26

Abbildung 3.2:  
Portscan des Containers *de-ondics-launchpad-db* aus einer anderen, auf *Nmap* basierenden App heraus.

einem entsprechenden Docker-Image<sup>14</sup> basiert. Die Struktur ist der *Anleitung zur App-Entwicklung* des ScaleIT-Wikis nachempfunden, allerdings wird der `entrypoint` des ursprünglichen Images ("*nmap*") mithilfe einer entsprechenden Zeile in der Datei `docker-compose.yml.tpl` überschrieben, da der Container ohne die üblicherweise über die Kommandozeile übergebenen Parameter sonst in keinen stabilen Zustand überführt werden kann:

```
version: '2'
services:
  de-aisec-nmap:
    image: uzyexe/nmap
    restart: always
    entrypoint: [ "nmap", "-Pn", "-sV", "-v",
      ↪ " 10.42.20.214" ]
```

Dies bewirkt zum einen die direkte Ausführung des in der `docker-compose.yml.tpl` definierten Scans, ermöglicht es zusätzlich jedoch auch, per *Execute Shell*-Funktion der *Rancher*-Weboberfläche weitere Scans zu starten. Bei der exemplarisch gescannten IP-Adresse handelt es sich um den Container *de-ondics-launchpad-db* der App *de-ondics-launchpad*. Wie in Abbildung 3.2 ersichtlich wird, ist es auf diesem Wege möglich, Port 3306 (MySQL) zu erreichen. Dieser Port sollte eigentlich jedoch nur innerhalb der App *de-ondics-launchpad* erreichbar sein.

Der Grund hierfür ist *Ranchers* flache Netzwerkstruktur. Alle Container befinden sich standardmäßig im selben managed-Netzwerk 10.42.0.0/16. Um

<sup>14</sup><https://hub.docker.com/r/uzyexe/nmap/>

zu verhindern, dass ein manipulierter Container potentiell alle anderen Container kompromittieren kann, ist deshalb ein zusätzlicher Schutz notwendig. *Rancher* bietet hierfür das Werkzeug der *Network Policy*<sup>15</sup> an, welches mithilfe des Infrastruktur-Services *network-policy-manager*, der über den offiziellen Rancher Catalog abrufbar ist, konfiguriert werden kann. Zur Auswahl stehen entweder die Konfiguration über das User-Interface (UI) oder aber per Application Programming Interface (API). Bei dem zu ändernden Parameter handelt es sich um *Everything Else* (UI) bzw. *defaultPolicyAction* (API), welches jeweils auf *deny* gestellt werden muss. Da sich die entsprechende Funktion der UI als eher unzuverlässig erwiesen hat, wird empfohlen die Konfiguration per API vorzunehmen. Zusätzlich wird empfohlen, *defaultPolicyAction* auf *deny* sowie bei der Konfiguration der API die im Folgenden aufgelisteten *Network Policy*-Regeln zu setzen, um die legitime Kommunikation zwischen verschiedenen Services / Containern innerhalb eines Stacks weiterhin zu gewährleisten:

```
{
  "within": "linked",
  "action": "allow"
}

{
  "within": "service",
  "action": "allow"
}

{
  "within": "stack",
  "action": "allow"
}
```

Wie in Abbildung 3.3 ersichtlich wird, läuft der Portscan der *Nmap*-App ins Leere, sobald die *Network Policy* korrekt konfiguriert ist.

### 3.3.2 Zulassen legitimer Kommunikation

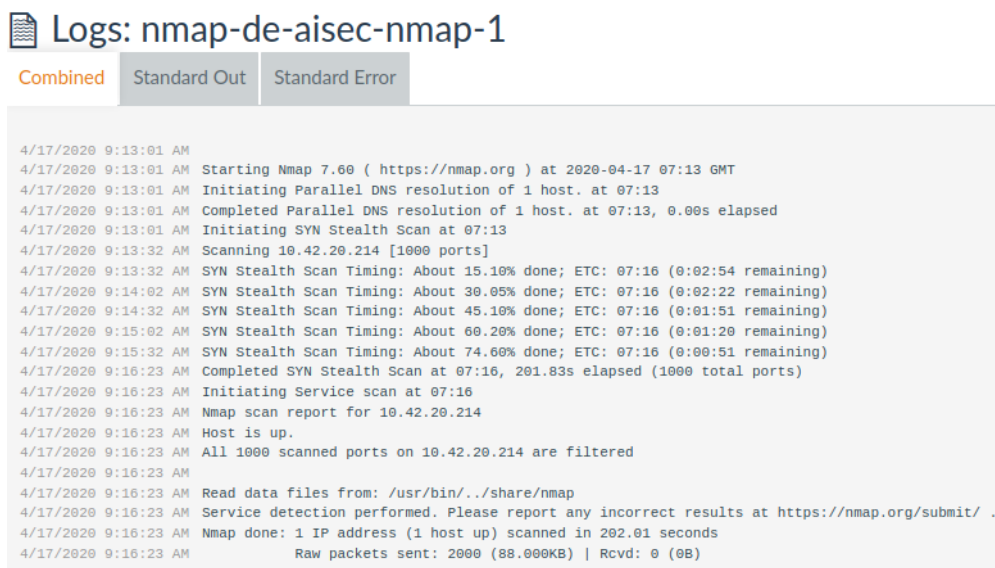
Um auch weiterhin legitime Kommunikation zwischen Services zu ermöglichen, die nicht zum selben Stack gehören, wie dies zum Beispiel beim *Launchpad* notwendig ist, können die entsprechenden Services dank der *linked/allow*-Regel anhand von *ServiceLinks*<sup>16,17</sup> direkt miteinander verlinkt werden, wie im

<sup>15</sup><https://rancher.com/docs/rancher/v1.6/en/rancher-services/network-policy/>

<sup>16</sup><https://rancher.com/docs/rancher/v1.1/en/api/v1/api-resources/serviceLink/>

<sup>17</sup><https://rancher.com/docs/rancher/v1.6/en/api/v2-beta/api-resources/loadBalancerService/#addservicelink>





```

Logs: nmap-de-aisec-nmap-1
Combined Standard Out Standard Error

4/17/2020 9:13:01 AM
4/17/2020 9:13:01 AM Starting Nmap 7.60 ( https://nmap.org ) at 2020-04-17 07:13 GMT
4/17/2020 9:13:01 AM Initiating Parallel DNS resolution of 1 host. at 07:13
4/17/2020 9:13:01 AM Completed Parallel DNS resolution of 1 host. at 07:13, 0.00s elapsed
4/17/2020 9:13:01 AM Initiating SYN Stealth Scan at 07:13
4/17/2020 9:13:32 AM Scanning 10.42.20.214 [1000 ports]
4/17/2020 9:13:32 AM SYN Stealth Scan Timing: About 15.10% done; ETC: 07:16 (0:02:54 remaining)
4/17/2020 9:14:02 AM SYN Stealth Scan Timing: About 30.05% done; ETC: 07:16 (0:02:22 remaining)
4/17/2020 9:14:32 AM SYN Stealth Scan Timing: About 45.10% done; ETC: 07:16 (0:01:51 remaining)
4/17/2020 9:15:02 AM SYN Stealth Scan Timing: About 60.20% done; ETC: 07:16 (0:01:20 remaining)
4/17/2020 9:15:32 AM SYN Stealth Scan Timing: About 74.60% done; ETC: 07:16 (0:00:51 remaining)
4/17/2020 9:16:23 AM Completed SYN Stealth Scan at 07:16, 201.83s elapsed (1000 total ports)
4/17/2020 9:16:23 AM Initiating Service scan at 07:16
4/17/2020 9:16:23 AM Nmap scan report for 10.42.20.214
4/17/2020 9:16:23 AM Host is up.
4/17/2020 9:16:23 AM All 1000 scanned ports on 10.42.20.214 are filtered
4/17/2020 9:16:23 AM
4/17/2020 9:16:23 AM Read data files from: /usr/bin/./share/nmap
4/17/2020 9:16:23 AM Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
4/17/2020 9:16:23 AM Nmap done: 1 IP address (1 host up) scanned in 202.01 seconds
4/17/2020 9:16:23 AM Raw packets sent: 2000 (88.000KB) | Rcvd: 0 (0B)

```

Abbildung 3.3:

Portscan des Containers *de-ondics-launchpad-db* aus einer anderen, auf *Nmap* basierenden App heraus mit restriktiver *Network Policy*-Konfiguration.

folgenden HTTP-Request anhand des Launchpads und ETCD exemplarisch darstellt:

```
HTTP/1.1 POST /v2-beta/projects/1a5/services/1s17/?
```

```
  ↪ action=addservicelink
```

```
Host: 10.0.3.30:8080
```

```
Accept: application/json
```

```
Content-Type: application/json
```

```
Content-Length: 69
```

```

{

  "serviceLink": {
    "name": "de-ondics-launchpad-app",
    "serviceId": "1s25"
  }
}

```

Alternativ ist es auch möglich, anhand von Labels<sup>18</sup> Gruppen von Services zu bilden, innerhalb derer diese Services jeweils miteinander kommunizieren können. Wichtig ist hierbei, dass die Gruppen nicht anhand der Bezeichnung des Labels, sondern des jeweils zugewiesenen Wertes gebildet werden. Das exemplarische Label `com.rancher.purpose` könnte also zum Beispiel zwei Services *a)* und *b)* verschiedener Stacks mit dem Wert `label: com.rancher.purpose`

<sup>18</sup><https://rancher.com/docs/rancher/v1.4/en/rancher-services/network-policy/>

= `registration` und zwei andere Services *c) und d)* ebenfalls verschiedener Stacks mit dem Wert `label: com.rancher.purpose = cross-stack-communication` gruppieren. Entsprechend könnten jeweils nur die Services *a) und b)* sowie *c) und d)* untereinander kommunizieren, nicht aber zum Beispiel *a) und d)*. Zwar haben diese das selbe Label zugewiesen bekommen, allerdings mit unterschiedlichen Werten. Die entsprechende, per API zu konfigurierende *Network Policy*-Regel würde lauten:

```
{
  "between": {
    "groupBy": "com.rancher.purpose"
  },
  "action": "allow"
}
```

### 3.3.3 Netzwerkzugriffsmöglichkeit durch Apps

Die bisher in Abschnitt 3.3 aufgeführten Erkenntnisse gelten unter der Annahme, dass Container *nicht* im privilegierten Modus ausgeführt werden. Das Ausführen im privilegierten Modus kann z. B. durch das Setzen des Parameters `privileged: true` in der jeweiligen `docker-compose.yml.tpl` einer App erfolgen und erlaubt es Containern, auch wenn die `defaultPolicyAction` der *Network Policy* auf `deny` gesetzt ist, mit jedem anderen Container zu kommunizieren. *ServiceLinks* oder die Zugehörigkeit zu bestimmten, per Label gebildeten Gruppen ist dafür nicht notwendig. Unter bestimmten Umständen kann sogar der Netzwerkverkehr zwischen Drittcontainern mitgehört werden. Exemplarisch wurde hierfür eine prototypische, privilegierte, auf Alpine Linux basierende App entwickelt, auf der die beiden folgenden Repos der Datei `/etc/apk/repositories` hinzugefügt wurden:

```
http://dl-cdn.alpinelinux.org/alpine/edge/testing
http://dl-cdn.alpinelinux.org/alpine/edge/main
```

Daraufhin konnte anhand der folgenden Befehlsabfolge das Netzwerk-Sniffing-Tool *Ettercap* installiert und im textbasierten (`-T`) Man-in-the-Middle-Modus unter Verwendung von Address Resolution Protocol (ARP) Poisoning (`-M arp`) gestartet werden:

```
# apk update
# apk upgrade
# apk add ethtool ettercap
# ettercap -T -M arp
```

Weiterhin ist es selbst aus nicht privilegierten Containern heraus, und wenn die `defaultPolicyAction` der *Network Policy* auf `deny` gesetzt ist, möglich,

dass Apps über das jeweils vor Ort bestehende Netzwerk mit erreichbaren Geräten bzw. Diensten kommunizieren und somit z. B. anhand von Portscans und Fingerprinting die Umgebung auskundschaften können. Es kann deshalb abhängig von den zu realisierenden Use Cases zur zusätzlichen Risikoreduzierung sinnvoll sein, darüber nachzudenken, ob Apps tatsächlich selbstständig Verbindungen initiieren müssen oder lediglich auf eingehende, vom Reverse Proxy weitergeleitete Anfragen reagieren sollen.

## 4 Leitfaden für Entwickler

Während die ersten Kapitel auf Bestandteile und Absicherung der ScaleIT Plattform eingehen, gibt Kapitel 4 Hinweise, wie Entwickler sichere Applikationen für die Plattform entwickeln können. Einen guten Einstieg bietet hier z. B. der Leitfaden zur Entwicklung sicherer Webanwendungen<sup>19</sup> des BSI.

### 4.1 Sicherer Entwicklungsprozess

Damit eine App sicher ist, ist es wichtig, den gesamten Lebenszyklus von der Anforderungsphase bis zum Ende des Supports abzusichern. Hierfür kann ein sicherer Entwicklungslebenszyklus (SDL) definiert werden, indem in den einzelnen Phasen bestimmte Maßnahmen umgesetzt werden.

- *Anforderungs- und Designphase*: Erstellen eines Sicherheitskonzepts, z. B. mithilfe einer Risikoanalyse. Hiermit kann festgestellt werden, welche Assets (Softwarekomponenten und Daten) schützenswert sind und wie exponiert diese Assets sind, also ob weitere Sicherheitsmaßnahmen benötigt werden, um das jeweilige Asset zu schützen. Ein ausgereiftes Sicherheitskonzept ist die Grundlage, um sichere Applikation implementieren zu können (Security by Design). Dies beinhaltet auch ein Datenschutz- sowie ein Rollen- und Berechtigungskonzept. Zur Absicherung der Kommunikation und Erreichbarkeit sollten die von ScaleIT bereitgestellten Mechanismen genutzt werden (siehe Kapitel 3). Auch wenn dies viele Angriffe bereits verhindern kann, sollte zusätzlich eine Härtung der Softwarekomponenten erfolgen (Defense in Depth).
- *Implementierungsphase*: Wenn das Sicherheitskonzept steht, kann mit der Implementierung begonnen werden. Auch hier können einige Prozesse helfen, um Schwachstellen und Sicherheitslücken von Anfang an zu vermeiden.
  - Erstellung und Umsetzung von Secure-Coding-Standards, wie in Abschnitt 4.3 beschrieben.
  - Sicherer Umgang mit Quellcode, wenn mehrere Entwickler daran arbeiten (z. B. Zugriffsregeln nach dem Least-Privilege-Prinzip zur Modifikation von Quellcode und Definition eines Change Managements).

---

<sup>19</sup>[https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Webanwendungen/Webanw\\_Auftragnehmer.pdf](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Webanwendungen/Webanw_Auftragnehmer.pdf)

- Führen einer aktuellen Liste von Common Vulnerabilities and Exposures<sup>20</sup> (CVEs) für genutzte Bibliotheken / Docker-Images.
- Benutzung von statischen Code-Analysern in der Entwicklungsumgebung.
- *Testphase*: Durchführen von statischen und dynamischen Security-Tests, sowohl von einzelnen Services als auch von der vollständigen App (Modultests, Funktionstests und Systemtests), siehe auch Abschnitt 4.4.
- *Betriebs- und Wartungsphase*:
  - Sichere Auslieferung: Signieren des Docker-Images und Ablage in vertrauenswürdigen Repository. Zudem sollte die Applikation mit einer sicheren Default-Konfiguration ausgeliefert werden. Zusätzliche Maßnahmen, die vom Benutzer getroffen werden müssen, sollten mindestens dokumentiert, am besten jedoch technisch erzwungen werden (wenn der Nutzer ein Passwort setzen muss z. B. durch eine sichere Passwort-Policy).
  - Sicheres Speichern von privaten Schlüsseln und Einbringen von Zertifikaten, wenn nötig. Dies sollte über einen Certificate Signing Request (CSR) erfolgen, damit der private Schlüssel nicht die Applikation verlassen muss.
  - Mitliefern einer Dokumentation, siehe auch Abschnitt 4.2.
  - Erstellen eines Software Security Incident Response Plans, indem ungeplante Störungen / Ausfälle dokumentiert werden.
  - Sicheres Patch-Management inklusive kontinuierlicher Beobachtung der CVEs von genutzten Bibliotheken / Docker-Images.
- *Deinstallation*: Bereitstellen von Funktionalitäten für das (sichere) Löschen von Daten.
- *Ende des Lebenszyklus / Supports*: Hinweis für den Nutzer, dass die Applikation nicht mehr gewartet wird.

### 4.2 Transparenz

Um einen sicheren Lebenszyklus einer App zu ermöglichen, ist Transparenz durch eine ausführliche Dokumentation unabdingbar. Dies ermöglicht es sowohl nachfolgenden Entwicklern als auch dem Ökosystembetreiber z. B. nachzuvollziehen, welche Schnittstellen aus welchem Grund geöffnet sind, welche API von außen angesprochen werden kann, ob sich schützenswerte Assets in der App befinden und auf welchen Bibliotheken / Docker-Images sie aufbaut. Letzteres

---

<sup>20</sup><https://cve.mitre.org/>

erlaubt es, nachzuverfolgen, ob in diesen Bibliotheken über die Zeit Verwundbarkeiten gefunden wurden und erste Schritte in Richtung eines sicheren Patch-Managements eingeleitet werden. Für Kommunikation nach außen muss klar ersichtlich sein, ob die App als Server oder als Client agiert und wo sich die Kommunikationsendpunkte befinden.

Auch ist es wichtig, dass man schützenswerte Assets nicht durch fehlende Dokumentation schützt (Security by Obscurity), sondern durch kryptographische Mechanismen. Private Schlüssel müssen selbstverständlich geschützt sein, der Mechanismus (Algorithmus, Schlüssellänge etc.) hingegen soll öffentlich sein. Nur so kann nachvollzogen werden, wann ein Mechanismus nicht mehr dem Stand der Technik entspricht und ersetzt werden muss.

### 4.3 Secure Coding

Bis zu 90 Prozent der Software Security Probleme werden durch Coding-Fehler verursacht.<sup>21</sup> Die *Common Weakness Enumeration*<sup>22</sup> listet Security Schwachstellen für Soft- und Hardware aufgeteilt in Kategorien wie Buffer Overflows, Cross-Site Scripting, SQL Injection etc. auf. Das *Open Web Application Security Project (OWASP)* bietet zahlreiche Informationen und Tooling, um Webapplikationen abzusichern, unter anderem eine Liste der zehn häufigsten Schwachstellen<sup>23</sup> und einen Secure-Coding-Leitfaden<sup>24</sup>.

Der Leitfaden kann als Basis für einen Coding-Standard für Webapplikation dienen. Allgemein bietet zum Beispiel der *BSI Grundschatz*<sup>25</sup> einen Standard, um eine gewisse Basis-Sicherheit umzusetzen. Die Computer Emergency Response Team (CERT)-Standards<sup>26</sup> werden ebenfalls häufig verwendet.

Im Folgenden sind einige Punkte herausgriffen, die auf jeden Fall beachtet werden sollten:

- Komplexität vermeiden / Minimalprinzip: Ein einfaches Design verhindert oft die meisten Schwachstellen.
- Nutzung von Standard-Bibliotheken für kryptographische Algorithmen: Diese sind überprüft und somit mit großer Wahrscheinlichkeit sicherer als Eigenimplementierungen.
- Minimieren und Absichern der Schnittstellen: Dies verringert die Angriffsfläche.

<sup>21</sup>[https://www.us-cert.gov/sites/default/files/publications/infosheet\\_SoftwareAssurance.pdf](https://www.us-cert.gov/sites/default/files/publications/infosheet_SoftwareAssurance.pdf)

<sup>22</sup><https://cwe.mitre.org/>

<sup>23</sup><https://owasp.org/www-project-top-ten/>

<sup>24</sup><https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/>

<sup>25</sup>[https://www.bsi.bund.de/DE/Themen/ITGrundschatz/ITGrundschatzStandards/ITGrundschatzStandards\\_node.html](https://www.bsi.bund.de/DE/Themen/ITGrundschatz/ITGrundschatzStandards/ITGrundschatzStandards_node.html)

<sup>26</sup><https://wiki.sei.cmu.edu/>

- Validierung von Eingangsdaten: Die offenen Schnittstellen sind im besten Fall nur von autorisierten Nutzern nutzbar. Dennoch sollten auch über diese Schnittstellen eingehende Daten überprüft werden, sodass darüber keine Schwachstellen ausgenutzt werden können (SQL Injection oder ähnliches).
- Sichere Fehlerbehandlung: Fehlerfälle sollten immer behandelt werden und in einen sicheren Zustand führen, sodass durch das Produzieren eines Fehlerfalls keine Sicherheitslücken entstehen.
- Default Deny: Die Nutzung von Whitelists sollte gegenüber Blacklists präferiert werden. Dies verhindert, dass neue oder nicht bedachte Fälle die Liste passieren können.
- Least Privilege: Es sollten immer so wenige Rechte wie möglich vergeben werden, um eine Ausnutzung der Rechte einzuschränken. Dies bedeutet insbesondere keine Nutzung des Privileged Mode in Docker Compose, da ansonsten ein kompromittierter Container Zugriff zum Host bekommen könnte.

#### 4.4 Security Testing

Security-Tests sollten sowohl während als auch nach der Implementierungsphase durchgeführt werden. Die ISO 29119 stellt eine Normenreihe dar, die Prozesse, Dokumentation, Techniken und Metriken für Software-Tests beschreibt und definiert. OWASP bietet zudem einen Security-Testing-Guide<sup>27</sup>.

Testen kann in statisches und dynamisches Testen aufgeteilt werden. Während statische Tests den Quellcode analysieren, arbeiten dynamische Tests auf dem Binary zur Laufzeit. Statische Tests können mithilfe von Code-Analysen, die Semantik und Syntax des Quellcodes untersuchen, (teil-)automatisiert werden. Statische Code-Analysen können in die Entwicklungsumgebung eingebunden werden und somit die Entwickler bereits während der Implementierung unterstützen, um Secure-Coding-Standards einzuhalten und Verwundbarkeiten wie Buffer Overflows zu vermeiden. Hierfür gibt es zahlreiche Tools für unterschiedliche Programmiersprachen<sup>28</sup>. Deutlich aufwendiger ist ein manueller Quellcode-Audit. Dieser kann allerdings viele Sicherheitslücken wie z. B. durch Logikfehler verursachte Schwachstellen finden, die ein automatisierter Test schwer erkennen kann. Auch hierfür bietet OWASP einen Leitfaden<sup>29</sup>.

Dynamische Tests können mithilfe von Verwundbarkeits-Scannern ebenfalls automatisiert ablaufen, auch hierfür gibt es einiges an Tooling<sup>30</sup>. Während die Scan-

---

<sup>27</sup><https://owasp.org/www-project-web-security-testing-guide/>

<sup>28</sup>[https://owasp.org/www-community/Source\\_Code\\_Analysis\\_Tools](https://owasp.org/www-community/Source_Code_Analysis_Tools)

<sup>29</sup><https://owasp.org/www-project-code-review-guide/>

<sup>30</sup>[https://owasp.org/www-community/Vulnerability\\_Scanning\\_Tools](https://owasp.org/www-community/Vulnerability_Scanning_Tools)

ner einfach zu verwenden sind und häufige Schwachstellen durchaus aufdecken können, können viele Sicherheitslücken jedoch nur durch das teilautomatisierte Fuzz-Testing (Testen mit ungültigen/unerwarteten Eingabewerten) oder mit manuellen Penetrationstests gefunden werden, welche jedoch Expertenwissen benötigen und deutlich aufwändiger sind.

Zudem können sich Sicherheitslücken bereits in genutzten Open-Source-Bibliotheken und Docker-Images von Drittanbietern befinden. Hierfür kann eine manuelle Suche erfolgen, eine umfangreiche Datenbank hierzu befindet sich auf <https://cve.mitre.org>, oder es können automatisierte Scanner dafür verwendet werden<sup>31</sup>.

## 4.5 Sichere Docker-Entwicklung

Zusätzlich zu den bereits genannten Hinweisen zur sicheren Entwicklung von Webanwendungen, gibt es einige Punkte, die speziell für Docker beachtet werden sollten. Während sich hauptsächlich der Ökosystembetreiber darum kümmern muss, dass Docker und Rancher richtig konfiguriert sind, gibt es ein paar Stellschrauben, mithilfe derer bereits der Entwickler dafür sorgen kann, dass seine Docker-Anwendung sicher ist.

Da bei der Erstellung einer auf Docker basierenden Anwendung fast immer auf bestehenden Images aufgebaut wird, ist es wichtig, dass diese Images als vertrauenswürdig angesehen werden, um sich damit nicht bereits Sicherheitslücken in die eigene Applikation zu holen. Hierfür gilt es, die bereits erwähnten Scanner und Datenbanken zu verwenden. Zudem sollten nur Images aus einer vertrauenswürdigen Docker-Registry verwendet werden. Um einer Manipulation auf der Kommunikationsstrecke zwischen Registry und Arbeitsplatz vorzubeugen, ist es empfehlenswert, signierte Images zu verwenden<sup>32</sup>.

In den Docker Docs sind zudem Best Practices aufgelistet, die helfen können, die Anwendung sicherer zu machen.<sup>33</sup> Ein mehrstufiges Bauen von Docker-Images kann vermeiden, dass Geheimnisse / private Schlüssel versehentlich veröffentlicht werden. Besser ist es jedoch, beispielsweise *Docker Secrets*<sup>34</sup> zur Verwaltung von privaten Schlüsseln zu verwenden. Es sollten nur Packages installiert werden, die auch benötigt werden, denn ein Container sollte genau ein Anliegen erfüllen und aus möglichst wenigen Layern bestehen – dies alles verringert die Komplexität. Eine eindeutige Versionierung unterstützt das Update Management. Anstatt von ADD sollte vorzugsweise die Instruktion COPY verwendet werden, da diese transparenter ist und über ADD zum Beispiel Remote-URLs eingebunden werden könnten, welche ebenfalls eine Quelle für Schwachstellen sein können.

<sup>31</sup><https://techbeacon.com/app-dev-testing/13-tools-checking-security-risk-open-source-dependencies>

<sup>32</sup><https://docs.docker.com/engine/security/trust/>

<sup>33</sup>[https://docs.docker.com/develop/develop-images/dockerfile\\_best-practices/](https://docs.docker.com/develop/develop-images/dockerfile_best-practices/)

<sup>34</sup><https://docs.docker.com/engine/swarm/secrets/>



## 4.6 Datenverschlüsselung

Vertrauliche Daten sollten nicht nur verschlüsselt übertragen, sondern auch verschlüsselt gespeichert werden. Bei Docker-Anwendungen werden diese Daten im Regelfall auf `Volumes` bzw. `Bind Mounts` abgelegt. Ist das System ausgeschaltet bzw. durch logische Zugriffskontrollmechanismen gesperrt, schützt eine (sehr ratsame) Vollverschlüsselung des Datenträgers bereits gegen unberechtigten Zugriff durch Dritte. Allerdings kann der Ökosystembetreiber nach wie vor auf diese Daten zugreifen. Normalerweise sollte es sich dabei zwar ohnehin um eine vertrauenswürdige Partei handeln, allerdings besteht im Falle einer Kompromittierung durch Dritte durchaus das Risiko eines sogenannten *Supply-Chain-Angriffs*, infolgedessen sich Angreifer alle Rechte des Ökosystembetreibers aneignen können.

Da der Ökosystembetreiber im Regelfall Vollzugriff auf das System sowie dessen Daten- und Arbeitsspeicher hat, ist es auch durch zusätzliche Verschlüsselung auf App-Ebene unter Benutzung von Mechanismen zur Schlüsselverteilung wie zum Beispiel *Docker Secrets* schwierig bis unmöglich, alle Daten vor seinem Zugriff zu schützen. Denn selbst wenn Daten auf diese Art und Weise verschlüsselt auf entfernten Speicherzielen, wie zum Beispiel in der Cloud, abgelegt werden, befinden sie sich, wie auch die entsprechenden Schlüssel, zumindest für eine Zeit lang unverschlüsselt im Arbeitsspeicher des Systems und können deshalb ausgelesen werden.

Abhilfe versucht hier das Projekt *SCONE*<sup>35</sup> zu schaffen, welches die vertrauenswürdige Laufzeitumgebung von Intels *Software Guard Extensions* (SGX) nutzt, um sensible Daten, wie zum Beispiel Schlüsselmaterial, auch vor Exfiltration durch privilegierte Nutzer zu schützen.<sup>36</sup> *SCONE* entstand zwar ursprünglich als ein akademisches Forschungsprojekt, bietet mittlerweile jedoch eine umfangreiche Auswahl an Images an, die direkt genutzt werden können. Unter anderem befindet sich auch das Secret Management System *Vault*<sup>37</sup> darunter.

Um sich gegen externe Angreifer zu schützen, kann es außerdem helfen, z. B. Datenbanken mit personenbezogenen Daten bereits auf Applikationsebene zu verschlüsseln, da die Vollverschlüsselung der Festplatte nicht gegen Angreifer schützt, die Sicherheitslücken in der App selbst finden und dadurch einzelne Daten unberechtigt auslesen können (beispielweise durch eine SQL Injection).

## 4.7 Produktschutz

Wenn das geistige Eigentum der Applikation schützenswert ist, sollten Sicherheitsmaßnahmen getroffen werden, um deren Extraktion zu verhindern oder

<sup>35</sup><https://scontain.com/>

<sup>36</sup><https://www.usenix.org/system/files/conference/osdi16/osdi16-arnautov.pdf>

<sup>37</sup><https://sconedocs.github.io/vault/>

zumindest zu erschweren. Eine Möglichkeit ist es, mittels *Obfuskation* schützenswerte Programmlogiken zu verschleiern, indem zum Beispiel im tatsächlich veröffentlichten Code Kommentare gelöscht werden, zusätzliche mathematische Berechnungen eingeführt werden oder mit komplizierten Tautologien der Quellcode um bedingte Anweisungen erweitert wird. Ausführliche Anweisungen hierfür bietet beispielsweise das Buch *Web Application Obfuscation*<sup>38</sup>.

Mit der experimentellen `containerd`-Erweiterung `imgcrypt`<sup>39,40</sup> kann zudem eine App bzw. der dazugehörige Container verschlüsselt werden, um die Programmlogik im Inneren während der Verteilung bzw. Bereitstellung der Container zu schützen.

---

<sup>38</sup><https://doc.lagout.org/security/Web%20Application%20Obfuscation/Web%20Application%20Obfuscation.pdf>

<sup>39</sup><https://developer.ibm.com/articles/encrypted-container-images-for-container-image-security-at-rest>

<sup>40</sup><https://github.com/containerd/imgcrypt/>

## 5 Leitfaden für Ökosystembetreiber

Der Ökosystembetreiber ist dafür verantwortlich, den Host abzusichern und Docker bzw. Rancher sicher zu konfigurieren. Daher ist es notwendig, die bestmögliche Sicherheit des zugrundeliegenden Betriebssystems *Ubuntu 16.04.06 LTS* sowie verschiedener darauf installierter Dienste und Anwendungen zu gewährleisten. Zudem sollte der Ökosystembetreiber Apps vor der Installation überprüfen/testen, insbesondere falls die Vertrauenswürdigkeit ihrer Entwickler nicht sichergestellt werden kann.

### 5.1 Automatische (Sicherheits-)Updates

In sämtlichen Arten von Software, einschließlich Betriebssystemen, werden regelmäßig neue Sicherheitslücken entdeckt. Ein zum Zeitpunkt der Installation sicheres System kann ohne zeitnahe und regelmäßige Sicherheitsupdates deshalb verwundbar gegen Angriffe werden. ScaleIT soll als einfach verwendbare Lösung eingesetzt werden, die es Entwicklern erlaubt, sich auf die eigentliche App-Entwicklung zu konzentrieren, ohne die Infrastruktur administrieren zu müssen, auf der das ScaleIT-Ökosystem aufbaut. Es ist deshalb empfehlenswert, von Haus aus automatische Sicherheitsupdates zu aktivieren, sodass *Ubuntu* sich auch ohne manuelles Zutun stets auf dem aktuellen Stand befindet.

Hierfür existieren unter anderem folgende Optionen<sup>41</sup>:

- Das `unattended-upgrades` Paket.
- Der Einsatz eines selbst entwickelten Update-Skripts, welches regelmäßig durch Ubuntu's Scheduler-Daemon `cron` ausgeführt wird.
- Das `cron-apt` Paket.

Aufgrund seiner Konfigurationsmöglichkeiten und einfacheren Bedienbarkeit wird im Folgenden auf die Automatisierung mithilfe des `unattended-upgrades` Pakets eingegangen. Hierfür wird das Paket im Regelfall zuerst installiert (bzw. aktualisiert, falls es bereits installiert ist):

```
$ sudo apt-get install unattended-upgrades
```

Anschließend wird `unattended-upgrades` anhand eines interaktiven Dialogs aktiviert, welcher durch folgenden Befehl aufgerufen wird:

---

<sup>41</sup><https://help.ubuntu.com/community/AutomaticSecurityUpdates>

```
$ sudo dpkg-reconfigure --priority=low unattended-
  ↪ upgrades
```

Wird die entsprechende Abfrage mit `Yes` beantwortet, erstellt die Routine automatisch die Datei `/etc/apt/apt.conf.d/20auto-upgrades`, welche wiederum von `cron` ausgelesen wird.

Standardmäßig ist in der Datei `/etc/apt/apt.conf.d/50unattended-upgrades` festgelegt, dass lediglich sicherheitsrelevante Updates automatisch installiert werden. Durch das Einkommentieren zusätzlicher Zeilen kann jedoch auch konfiguriert werden, sämtliche verfügbaren Softwareaktualisierungen automatisch zu installieren. Da Updates potentiell zu Konflikten führen können, die die Verfügbarkeit der installierten Dienste beeinträchtigen könnten, wird vor einer Aktivierung dieser Funktion für alle Pakete empfohlen, entsprechende Abwägungen bzgl. kritischer Dienste bzw. Pakete zu treffen.

## 5.2 Vollverschlüsselung von Datenträgern

Die Vollverschlüsselung des Datenträgers (Full Disk Encryption / FDE) soll die Vertraulichkeit der sich auf dem System befindlichen Daten einschließlich Apps und deren Nutzdaten im Falle von physischen Zugriffen durch Unbefugte schützen. Während Zugriffskontrollen wie Login-Daten nur das laufende System schützen, verhindert die Festplattenverschlüsselung, dass der Datenträger ausgebaut wird und an einem anderen System, das unter der Kontrolle des Angreifers steht, ausgelesen werden kann. Dies ist insbesondere notwendig, wenn die Hardware in einem nicht optimal geschützten Umfeld, wie zum Beispiel einer Baustelle, betrieben wird.

Innerhalb der Linux-Betriebssystemfamilie, zu der auch das von ScaleIT eingesetzte Ubuntu gehört, ist *Linux Unified Key Setup* (LUKS) bzw. `cryptsetup` das gängigste Verfahren, um Datenträger zu verschlüsseln. So wird es von den meisten Distributionen bei aktivierter Option zur Verschlüsselung des Betriebssystems bereits automatisch während dessen Installation verwendet. Der voreingestellte Verschlüsselungsalgorithmus *Advanced Encryption Standard* im Betriebsmodus XTS (`aes-xts-plain64`) gilt unter Verwendung einer ausreichend langen Schlüssellänge<sup>42</sup> nach heutiger Erkenntnis als sicher und sollte deshalb beibehalten werden. Die im Vergleich zu anderen AES-Implementierungen große Schlüssellänge rührt daher, dass im Betriebsmodus XTS der Schlüssel intern geteilt und die beiden Teilschlüssel für unterschiedliche Zwecke verwendet werden.

Die nachträgliche Verschlüsselung eines unverschlüsselten und bereits produktiven Systems ist aufgrund des hohen Risikos von Datenverlust nicht empfehlenswert. Stattdessen wird empfohlen, ein neues, direkt während der Installation zu verschlüsselndes System aufzusetzen und Produktivdaten auf dieses zu migrieren.

<sup>42</sup>Die minimale Schlüssellänge sollte bei 256 Bit liegen, wobei die Standardschlüssellänge ab Ubuntu 19.04 512 Bit beträgt.

Diese Vorgehensweise erlaubt ausgiebiges Testen sowie einen nahtlosen Übergang vom alten zum neuen System.

Auf eine Anleitung zum nachträglichen Verschlüsseln von Systemen wird an dieser Stelle aufgrund der eben genannten Gründe verzichtet. Auch die herkömmliche, bereits während der Installation aktivierte FDE ist aufgrund des üblicherweise benutzerfreundlichen Installationsdialogs selbsterklärend. Nichtsdestotrotz werden im Folgenden einige wichtige Details behandelt.

Bei der Verschlüsselung ist darauf zu achten, eine ausreichend lange, geheimzuhaltende Passphrase zu wählen. Da die sonst ratsame Verwendung eines sicheren Passwortmanagers in Kombination mit einem möglichst zufälligen Passwort bei der lokalen Entschlüsselung<sup>43</sup> eines Systems nicht praktikabel ist, da die Möglichkeit des *Copy + Paste* fehlt, wird die Verkettung mehrerer Worte in Kombination mit Zahlen und Sonderzeichen empfohlen. Worte, die in Zusammenhang mit dem System / Projekt stehen und deshalb leicht zu erraten wären, sollten vermieden werden. Diese Vorgehensweise hat den Vorteil, dass sich Menschen trotz ausreichender Länge an solche Passphrasen leichter erinnern können und sie auch beim Einsatz von Passwortmanagern auf anderen System leicht manuell übertragbar sind.

Obwohl das Verfahren landläufig als *Vollverschlüsselung* bezeichnet wird, wird in der Praxis nichtsdestotrotz nicht der gesamte Datenträger verschlüsselt. Die verschiedenen Betriebssystem- (/) und Datenpartitionen<sup>44</sup> (/home) werden zwar in der Tat verschlüsselt. Die Bootpartition (/boot) muss jedoch unverschlüsselt bleiben, damit das System den initialen Bootloader<sup>45</sup> laden kann, welcher wiederum für die Entschlüsselung zuständig ist.

Um diesen unverschlüsselten Teil des Systems gegen Manipulation zu schützen, empfiehlt sich der Einsatz von *Secure Boot*.<sup>46</sup> Hierfür ist die Verwendung des *Unified Extensible Firmware Interface* (UEFI) notwendig. *Secure Boot* erlaubt es (vereinfacht formuliert), eine kryptografisch signierte Version von GRUB zu validieren und bei gültiger Signatur zu starten. Die Einschleusung von Schad- bzw. Spähsoftware kann dadurch verhindert bzw. wesentlich erschwert werden.

Es ist abschließend anzumerken, dass der Schutz bei nicht ausgeschalteten, sondern lediglich durch logische Zugriffskontrollen gesperrten System dadurch gefährdet ist, dass Daten (unter anderem Schlüsselmaterial für die Vollverschlüsselung) im Arbeitsspeicher nach wie vor im Klartext vorliegen und durch diverse Angriffe ausgelesen werden können. Zusammen mit der TU München arbeitet das Fraunhofer AISEC an prototypischen Lösungen für dieses Problem.<sup>47</sup>

---

<sup>43</sup>Es besteht die Möglichkeit, ein verschlüsseltes System per SSH aus der Ferne zu entschlüsseln: [https://www.thomas-krenn.com/de/wiki/Voll-verschl%C3%BCssertes-System\\_via\\_SSH\\_freischalten](https://www.thomas-krenn.com/de/wiki/Voll-verschl%C3%BCssertes-System_via_SSH_freischalten)

<sup>44</sup>Abhängig von der gewählten Konstellation können diese auch auf einer Partition liegen.

<sup>45</sup>Im Fall von Ubuntu zumeist der *Grand Unified Bootloader* (GRUB), es gibt jedoch auch Alternativen, wie z. B. *Syslinux*.

<sup>46</sup><https://wiki.ubuntu.com/UEFI/SecureBoot>

<sup>47</sup><https://dl.acm.org/doi/10.1145/3374664.3375747>

### 5.3 Allgemeine Reduzierung der Angriffsfläche

Aufgrund der bereits beschriebenen Problematik, dass für sämtliche Software regelmäßig Sicherheitslücken entdeckt, diese jedoch nicht immer zeitnah geschlossen werden können, birgt jede auf dem System ausgeführte bzw. sogar lediglich installierte Software ein inhärentes Sicherheitsrisiko in sich. Trotz der regelmäßigen Installation von Sicherheitsupdates für Betriebssystem sowie Dienstprogramme ist es deshalb empfehlenswert, nicht benötigte Software auf dem Hostsystem zu deaktivieren bzw. im Idealfall komplett zu deinstallieren. Obwohl jede Art von installierter Software ein potentiell Sicherheitsrisiko darstellt, sollte das initiale Hauptaugenmerk auf der Reduzierung von Diensten liegen, die von außerhalb des Systems erreichbar sind. Ein Beispiel hierfür wäre z. B. der native SSH-Dienst in einem Szenario, in dem für die ScaleIT-Instanz ohnehin keine Internet-Konnektivität, sondern eine ausschließliche Wartung vor Ort direkt am Gerät vorgesehen ist.

Software sollte im Auslieferungszustand idealerweise eine sichere Konfiguration vorweisen. Leider ist das unter anderem aus Gründen der Kompatibilität und Funktionalität nicht immer der Fall. Entsprechend ist es wichtig, diese Konfiguration nach der Installation entsprechender Software und Systeme selbst vorzunehmen.

### 5.4 Transport Layer Security

*Transport Layer Security* (TLS) ist der De-Facto-Standard zur Realisierung sicherer Kommunikation über das Internet, welcher unter anderem auch in Protokollen wie dem *Hypertext Transfer Protocol Secure* (HTTPS) eingesetzt wird. 2018 erschien die aktuelle Version 1.3 der entsprechenden TLS-Spezifikation.<sup>48</sup> Die veralteten Versionen 1.0 und 1.1 gelten wegen ihrer Verwendung der als unsicher geltenden Hashfunktion *SHA-1* ebenfalls als unsicher und sollten laut BSI nicht mehr verwendet werden. Für die Versionen 1.2 und 1.3 existiert von Seiten des BSI eine Liste empfohlener Cipher-Suites, deren ausschließliche Nutzung in der jeweiligen TLS Implementierung explizit konfiguriert werden sollte.<sup>49</sup> Zusätzlich sollte regelmäßig überprüft werden, ob bereits aktuellere Empfehlungen seitens des BSI vorliegen, vor allem vor dem Hintergrund der Migration zu sogenannter *Post-Quanten-Kryptografie*.<sup>50</sup> Die hinter dem bekannten Webbrowser *Firefox* stehende *Mozilla Foundation* stellt einen Generator bereit, welcher es erlaubt, für verschiedene Webserver automatisch eine entsprechende Konfiguration zu

<sup>48</sup><https://tools.ietf.org/html/rfc8446>

<sup>49</sup>[https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2.pdf?\\_\\_blob=publicationFile&v=7](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2.pdf?__blob=publicationFile&v=7)

<sup>50</sup>[https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Krypto/Post-Quanten-Kryptografie.pdf?\\_\\_blob=publicationFile&v=2](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Krypto/Post-Quanten-Kryptografie.pdf?__blob=publicationFile&v=2)

erzeugen, welche im Nachgang jedoch noch einmal auf die Konformität mit den BSI-Empfehlungen hin überprüft werden sollte.<sup>51</sup>

Aus Kompatibilitätsgründen unterstützen viele Systeme jedoch auch weiterhin alte Versionen, von deren Verwendung eigentlich abgeraten wird. Diese Abwärtskompatibilität ermöglicht es Angreifern, sogenannte *Downgrade Attacks*<sup>52</sup> durchzuführen, welche die Sicherheitsstandards der Kommunikation zwischen Client und Server verringern und damit wiederum leichter angreifbar machen.

Neben der Verwendung von sicheren Cipher-Suites gibt es bei der sicheren Implementierung von TLS noch viele weitere Details zu beachten, welche den Rahmen dieser Arbeit sprengen würden. Es wird an dieser Stelle deshalb auf die sehr ausführlichen *SSL and TLS Deployment Best Practices* der *Qualys SSL Labs* verwiesen.<sup>53</sup>

## 5.5 Secure Shell

Um sich per SSH auf das ScaleIT-System schalten zu können, läuft dort der *SSH Daemon* (*sshd*), welcher auf Port 22 auf eingehende Verbindungen wartet. Damit das grundsätzlich eigentlich als sicher geltende Protokoll die Sicherheit des Systems als Ganzes nicht in unnötig hohem Maße gefährdet, sollten bei der Konfiguration von *sshd* in der Datei `/etc/ssh/sshd_config` einige von der Standardkonfiguration abweichende Details beachtet werden:

- `AllowUsers`: Anstatt allen lokalen Benutzern automatisch Zugriff per SSH zu gewähren, kann dies hier durch Angabe von durch Leerzeichen separierte Benutzernamen explizit und granular erfolgen.
- `Ciphers`: Analog zu den Empfehlungen in Abschnitt 5.4 sollten auch für SSH ausschließlich vom BSI empfohlene Cipher-Suites zur Verschlüsselung verwendet werden.<sup>54</sup> Sie werden jeweils durch ein Komma voneinander separiert.
- `HostKeyAlgorithms`: Die vom BSI empfohlenen Verfahren zur Server-Authentisierung können anhand des Parameters `HostKeyAlgorithms` konfiguriert werden.
- `KexAlgorithms`: Die vom BSI empfohlenen Verfahren zur Schlüsseleignung können anhand des Parameters `KexAlgorithms` konfiguriert werden.

---

<sup>51</sup><https://ssl-config.mozilla.org/#server=nginx&version=1.17.7&config=modern&openssl=1.1.1d&guideline=5.4>

<sup>52</sup>[https://link.springer.com/chapter/10.1007/978-3-030-01704-0\\_27](https://link.springer.com/chapter/10.1007/978-3-030-01704-0_27)

<sup>53</sup><https://github.com/ssllabs/research/wiki/SSL-and-TLS%20Deployment-Best-Practices>

<sup>54</sup>[https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-4.pdf?\\_\\_blob=publicationFile](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-4.pdf?__blob=publicationFile)

- `MACs`: Die vom BSI empfohlenen Verfahren zur MAC-Sicherung können anhand des Parameters `MACs` konfiguriert werden.
- `PasswordAuthentication`: Auch dieser Parameter sollte grundsätzlich auf `no` stehen. Statt Passwörtern sollten öffentliche Schlüssel zur Authentisierung verwendet werden, da diese wesentlich resistenter gegen Brute-Force-Angriffe sind.
- `PermitRootLogin`: Sollte grundsätzlich auf `no` gesetzt werden. Der Zugriff auf den `root`-Account ist nach wie vor über den Umweg eines anderen Benutzers möglich. Da der `root`-Account jedoch auf jedem Linux-System existiert, würde es potentiellen Angreifern eine Brute-Force Attacke erleichtern, wenn sie sich direkt als `root` authentisieren könnten.
- `X11Forwarding`: Da zur Server-Administration üblicherweise das Terminal benutzt wird, sollte X11-Forwarding, also die Möglichkeit zur grafischen Benutzerführung, durch setzen auf `no` deaktiviert werden. Diese Maßnahme reduziert Angriffsfläche in Form des Rückkanals vom Server zum Client, der durch aktiviertes X11-Forwarding aufgebaut werden kann.

Wenn der SSH-Zugang lediglich aus fest definierten Netzwerksegmenten heraus bzw. über eine fest definierte IP-Adresse aufgerufen wird (zum Beispiel die statische externe IP-Adresse des Ökosystembetreibers; hier beispielhaft `xxx.xxx.xxx.xxx`), kann es sinnvoll sein, den Zugang durch eine entsprechende `iptables`-Regel zusätzlich zu limitieren. Dies gilt insbesondere, wenn für die Authentisierung nicht - wie oben empfohlen - öffentliche Schlüssel, sondern stattdessen Passwörter verwendet werden:

```
$ sudo iptables -A INPUT -p tcp --dport 22 --source xxx.
↪ xxx.xxx.xxx -j ACCEPT
$ sudo iptables -A INPUT -p tcp --dport 22 -j DROP
```

Falls der Zugriff aufgrund operativer Umstände aus unterschiedlichen, schwer vorhersehbaren Netzwerksegmenten bzw. von verschiedenen IP-Adressen erfolgen muss, kann alternativ auch das flexiblere `denyhosts` verwendet werden:

```
$ sudo apt-get install denyhosts
```

Sollen IP-Adressen bei mehrmalig fehlerhaften Login-Versuchen nicht dauerhaft gesperrt bleiben, muss in der Konfigurationsdatei `/etc/denyhosts.conf` der standardmäßig leere Parameter `PURGE_DENY` nach der Installation auf einen konkreten Zeitraum festgelegt werden (zum Beispiel `1d` für einen Tag oder `1w` für eine Woche). Für den Fall, dass der Zugriff trotz der überwiegenden Verwendung dynamischer IP-Adressen auch von einer statischen IP-Adresse möglich ist, sollte diese in die Datei `/etc/hosts.allow` eingetragen werden, um im Falle einer irrtümlichen Sperrung immer eine Möglichkeit zu haben, diese wieder aufzuheben.

Die Anwendung von `iptables` oder `denyhosts` ersetzt jedoch keineswegs notwendige Infrastrukturmaßnahmen zur Netzwerksicherheit, wie zum Beispiel



Firewalls und Virtual Private Networks (VPNs) sowie deren korrekte und konsistente Konfiguration.

Sollte der SSH-Dienst allgemein nicht benötigt werden, gilt der in Abschnitt 5.3 formulierte Grundsatz. Der Dienst sollte also gestoppt und deinstalliert werden:

```
$ sudo systemctl stop ssh.service
$ sudo systemctl disable ssh.service
$ sudo apt-get remove openssh-server
```

## 5.6 Antivirensoftware

Obwohl geeignete Sicherheitsvorkehrungen die Wahrscheinlichkeit verringern, kann es nach wie vor zu Infektionen durch Schadsoftware kommen. Um diese bereits in einem frühen Stadium erkennen und gegensteuern zu können, wird empfohlen, das System regelmäßig mit Antivirensoftware zu scannen und diese auf dem aktuellen Stand zu halten. Ubuntu hält in seinen Paketquellen dafür zum Beispiel die Software *ClamAV*<sup>55</sup> bereit:

```
$ sudo apt-get install clamav clamav-daemon clamav-
↳ freshclam
```

Nach der Installation müssen die Signaturen für Schadsoftware manuell aktualisiert werden:

```
$ sudo freshclam
```

Alternativ besteht die Möglichkeit, *freshclam* als Dienst zu starten, welcher in der Standardkonfiguration jede Stunde automatisch nach Updates sucht:

```
$ sudo freshclam -d
$ sudo service clamav-freshclam start
```

ClamAV kann in den Betriebsmodi *On-Access*<sup>56</sup> oder *One-Time Scanning*<sup>57</sup> ausgeführt werden. Bei *On-Access* werden Dateien gescannt, wenn jeweils auf sie zugegriffen wird, *One-Time Scanning* erlaubt das einmalige Scannen bestimmter Dateien, ganzer Pfade oder des vollständigen Systems. *One-Time Scanning* kann unter Verwendung eines Cronjobs automatisiert und zum Beispiel einmal am Tag ausgeführt werden:

```
$ sudo cat > /etc/cron.daily/dailyclamscan.sh <<EOF
#!/bin/bash
clamscan -ir --log=/home/$USER/clamscan.log /
EOF
$ sudo chmod +x /etc/cron.daily/dailyclamscan.sh
```

<sup>55</sup><https://wiki.ubuntuusers.de/ClamAV/>

<sup>56</sup><https://www.clamav.net/documents/on-access-scanning>

<sup>57</sup><https://www.clamav.net/documents/scanning>

Das Scannen von Dateien, die von Docker benutzt werden, kann zu Problemen führen.<sup>58</sup> Die entsprechenden Dateien können deshalb entweder explizit vom Virenskan ausgeschlossen oder aber die Docker-Container für die Zeit des Scans kontrolliert heruntergefahren werden. Aus Perspektive der IT-Sicherheit ist letztgenannte Alternative zu bevorzugen, da dabei auch die mit Docker in Verbindung stehenden Daten auf potentiellen Schadcode hin überprüft werden können. Welche der beiden Optionen letztendlich verwendet wird, muss jedoch im Einzelfall anhand der Verfügbarkeits-Anforderungen der entsprechenden Docker-Container entschieden werden.

## 5.7 Docker Konfiguration

Obwohl dies mit einigen operativen Einschränkungen verbunden ist, sollten Docker bzw. die von dem Dienst kontrollierten Container nicht als privilegierter Nutzer `root` oder unter Zuhilfenahme des Befehls `sudo` gestartet werden. Dies könnte im Falle einer Schwachstelle in Docker selbst oder einer kompromittierten Anwendung innerhalb eines Docker-Containers die Kompromittierung des gesamten Systems begünstigen. Stattdessen sollte Docker im sogenannten *Rootless Mode*<sup>59</sup> gestartet werden. Um einem unprivilegierten Nutzer zu ermöglichen, Docker im Rootless Mode zu starten, muss er sich neu anmelden nachdem er zur Nutzergruppe `docker` hinzugefügt wurde:

```
$ sudo usermod -aG docker $USER
```

Um den Zugriff auf Ressourcen des Betriebssystems aus potentiell kompromittierten Containern heraus weiter zu erschweren, können außerdem Sicherheitsfeatures des Linux-Betriebssystems, wie z. B. *AppArmor*<sup>60</sup> oder *seccomp*<sup>61</sup> eingesetzt werden.

## 5.8 Backups

Datenverlust kann verschiedenste Ursachen haben und kündigt sich in den seltensten Fällen an:

- Abnutzung von Hardware;
- Konfigurationsfehler;
- Stromausfall;
- Feuer- oder Wasserschaden;

<sup>58</sup><https://docs.docker.com/engine/security/antivirus/>

<sup>59</sup><https://docs.docker.com/engine/security/rootless/>

<sup>60</sup><https://docs.docker.com/engine/security/apparmor/>

<sup>61</sup><https://docs.docker.com/engine/security/seccomp/>

- Diebstahl;
- Befall durch Schadsoftware, insbesondere *Ransomware* und *Wiper*.

Um im Fall eines Datenverlustes vorbereitet zu sein, ist es notwendig, die Kritikalität der auf einem System gespeicherten Daten im Vorfeld zu definieren und kritische Daten regelmäßig zu sichern. Um Daten vor physischen Bedrohungen wie Feuer, Wasser und Diebstahl zu schützen, ist es notwendig, dass die Sicherungskopien räumlich entfernt gelagert werden. Im Hinblick auf Ransomware bieten sich vor allem vor nachträglicher Manipulation geschützte Offline-Backups an.

Nutzdaten, welche durch Anwendungen erzeugt werden, die innerhalb eines Docker-Containers ausgeführt werden, werden in der Regel auf Docker-Volumes unter dem Pfad `/var/lib/docker/volumes/` gespeichert. Um diese Daten zu sichern ist es ausreichend, die entsprechenden Verzeichnisse regelmäßig (zum Beispiel mithilfe eines Cronjobs) auf eine oder idealerweise mehrere sichere Destination(en) zu sichern.

## 5.9 Security Testing

Um sicherheitskritische Zustände von Apps zu vermeiden, sind Ökosystembetreiber angehalten, den Download von Images ausschließlich von vertrauenswürdigen Registries zuzulassen. Sollten Entwickler in ihren Apps selbsterstellte Images einbinden, sollten diese signiert sein.<sup>62</sup> Außerdem sind entsprechende Konfigurationen vor Inbetriebnahme einer neuen App zu überprüfen. Hierfür können z. B. sowohl auf App- als auch auf Infrastruktur-Ebene Kataloge sicherheitsrelevanter Konfigurationen erstellt sowie regelmäßig aktualisiert werden. Potentiell gefährliche Systemzustände können zum Beispiel durch das privilegierte Ausführen von Containern herbeigeführt werden. Das kann zum einen durch das in Abschnitt 5.7 beschriebene, privilegierte Ausführen des Docker-Dienstes selbst als `root` bzw. unter Benutzung des Befehls `sudo`, aber beispielsweise auch durch die Verwendung des Parameters `privileged: true` in der jeweiligen `docker-compose.yml.tpl` einer einzelnen App geschehen. Privilegierte Container können z. B. im Rahmen erweiterter Zugriffsrechte auf das CNI, wie sie in Unterabschnitt 3.3.3 beschrieben werden, ein erhöhtes Risiko für Missbrauch bergen. Weitere Beispiele sind die Verwendung unsicherer Komponenten bzw. Docker-Images innerhalb einer App oder die unbeabsichtigte Freigabe von Ports (durch das Stichwort `EXPOSE` in Docker Compose erkennbar).

Neben der manuellen Prüfung durch den Ökosystembetreiber ist hier auch eine gewisse Automatisierung möglich. In der Docker-Community sind verschiedene Arten von Security- bzw. Vulnerability-Scannern verbreitet:

---

<sup>62</sup>[https://docs.docker.com/engine/security/trust/content\\_trust/#signing-images-with-docker-content-trust](https://docs.docker.com/engine/security/trust/content_trust/#signing-images-with-docker-content-trust)

- Zum einen sind das Scanner, welche die Docker-Installation selbst untersuchen. Ein Beispiel hierfür sind die im Rancher Community Katalog abrufbare App *Rancher Bench Security*<sup>63</sup> und *dockscan*<sup>64</sup>.
- Zum anderen existieren Scanner, welche die verwendeten Komponenten innerhalb eines Containers auf potentielle Schwachstellen scannen. Zu dieser Kategorie gehört zum Beispiel *Clair*<sup>65</sup>. Eine direkt in Rancher nutzbare Version von *Clair* findet sich im Rancher Katalog der *European Environment Agency*<sup>66</sup>.

Um die Erkennungsrate potentieller Schwachstellen der eigenen Docker- bzw. Rancher-Installation des Ökosystembetreibers als auch der installierten, durch Drittanbieter entwickelten Apps zu erhöhen, ist es sinnvoll, jeweils mindestens einen Scanner beider Arten zu nutzen.

Außerdem bietet es sich an, die im bisherigen Dokument beschriebenen defensiven Sicherheitsmaßnahmen durch ausgewählte offensive Maßnahmen zu ergänzen und somit zum Beispiel Fehlkonfigurationen, die von Angreifern ausgenutzt werden können, aktiv zu entdecken. Der Fokus sollte hierbei nicht nur auf dem ScaleIT-Ökosystem, sondern auf dem Gesamtsystem liegen, das heißt auch auf der Kommunikation des Hosts und der einzelnen Container mit verschiedenen anderen Komponenten.

---

<sup>63</sup><https://github.com/rancher/community-catalog/tree/master/templates/rancher-bench-security>

<sup>64</sup><https://github.com/kost/dockscan>

<sup>65</sup><https://github.com/quay/clair>

<sup>66</sup><https://github.com/eea/eea.rancher.catalog>

## 6 Fazit

Die Architektur der ScaleIT-Plattform ist bewusst funktional und offen gehalten, um Entwicklern den Umgang mit den verschiedenen Komponenten so leicht wie möglich zu machen. Diese Offenheit birgt jedoch Risiken. Auch wenn die ScaleIT-Instanz nicht direkt mit dem Internet verbunden ist, existieren, wie eingangs erwähnt und in Abbildung 1.1 aufgezeigt, verschiedene Angriffsvektoren, die die Kompromittierung der gesamten Plattform oder ihrer Komponenten ermöglichen könnten.

Einige Aspekte setzen dabei bereits bei der grundlegenden Architektur an. So wäre es z. B. empfehlenswert, die der ScaleIT CE zugrunde liegende Linux-Distribution *Ubuntu 16.04 LTS*, deren offizieller Support (zumindest ohne eine separat abzuschließende *Extended Security Maintenance*) Mitte 2020 endet<sup>67</sup>, durch eine aktuelle Version zu ersetzen. Besonders würde sich hier z. B. die im April 2020 erschienene und somit bis Mitte 2024 unterstützte *Long Time Support (LTS)*-Version 20.04 anbieten. Auch die verwendete Version 1.6.27 von Rancher erreicht Mitte 2020 das Ende ihres Lebenszyklus<sup>68</sup> und erhält ab Juli 2020 keine sicherheitsrelevanten Updates mehr.

Nichtsdestotrotz bietet ScaleIT CE in der vorliegenden Form ein Repertoire an verfügbaren Sicherheitsmechanismen, die allerdings zum Teil noch umgesetzt werden müssen. Es ist deshalb empfehlenswert, so viele dieser Sicherheitsmechanismen wie möglich standardmäßig zu aktivieren und gleichzeitig Entwickler sowie Ökosystembetreiber im Umgang mit diesen Mechanismen zu schulen. Dadurch wird nicht nur allgemein die bessere Handhabung der verschiedenen Tools gefördert, sondern beiden Gruppen durch ihr gesteigertes Sicherheitsbewusstsein auch die Möglichkeit gegeben, Chancen und Risiken der Plattform selbstständig abzuwägen sowie diese Abwägungen bewusst in die Konfiguration einzelner Apps bzw. der gesamten Plattform einfließen zu lassen.

Auch wenn die vollumfängliche Betrachtung, Bewertung und Optimierung aller verwendeten Komponenten im Rahmen dieses Leitfadens nicht möglich ist, soll er doch einen wesentlichen Teil beitragen. Kapitel 1, Kapitel 2 und Kapitel 3 tragen durch die Einführung verschiedener Komponenten sowie deren Risiken zum allgemeinen Verständnis der Plattform bei. Kapitel 4 und Kapitel 5 bieten Entwicklern und Ökosystembetreibern weiterhin konkrete Handlungsanweisungen, um das erlangte Sicherheitsbewusstsein in die Praxis umzusetzen.

---

<sup>67</sup><https://ubuntu.com/about/release-cycle>

<sup>68</sup><https://rancher.com/support-maintenance-terms/>

