Sichere Systeme: Sicheres Heimmonitoring (SichHeimMonitor)

Grodon, Placht, Russwurm
Hochschule für Angewandte Wissenschaften München
Fakultät für Informatik und Mathematik

15. Januar 2016

Inhaltsverzeichnis

1 Glossar

Der folgende Abschnitt liefert eine Einführung in die verwendeten Programme und Technologien.

1.1 Docker

Docker ist ein in der Programmiersprache Go entwickeltes Open Source Projekt, welches seit 2013 existiert. Es hat sich zum Ziel gesetzt, die automatisierte Bereitstellung von Applikationen innerhalb von Software Containern zu ermöglichen. Dazu stellt Docker eine Abstraktionsebene zur Verfügung um auf bestehende Funktionalität des Linux Kernel zuzugreifen. Zu den Funktionen gehören cgroups (Control Groups) und kernel-namespace.
Cgroups ermöglichen es, Ressourcen für eine Sammlung von Prozessen zu limitieren, verwalten und isolieren. Bei den Ressourcen handelt es sich beispielsweise um die verwendete CPU Leistung, I/O Zugriffe oder Netzwerk Ressourcen.
Kernel Namespace fassen eine Gruppe von Prozessen zu einer Einheit zusammen. Die Kommunikation ist nur mit dem Elternprozess und untereinander möglich.
Diese und weitere Funktionen erlauben es, getrennte Container, innerhalb eines gemeinsamen Host zu betreiben, ohne den Overhead einer Virtuellen Maschine. [1]

1.1.1 Unterschied zwischen Docker und Virtuellen Maschinen

Abbildung 1 zeigt exemplarisch die Architektur eines Systems bei der Verwendung einer oder mehrere virtueller Maschinen.

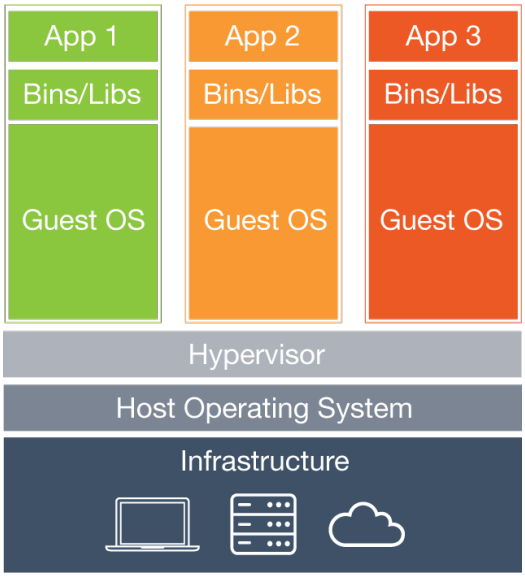


Abbildung 1: Überblick Virtuelle Maschinen [(1)]

Jede virtuelle Maschine enthält das Gast Betriebssystem, eine oder mehrere Anwendungen und die dazugehörigen Abhängigkeiten und Bibliotheken.
Im Vergleich dazu zeigt Abbildung 2 exemplarisch die Architektur eines Systems bei der Verwendung von Docker.

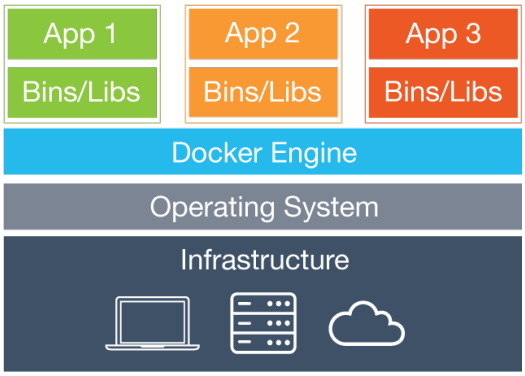


Abbildung 2: Überblick Virtuelle Maschinen [(2)]

[(1)] `https://www.docker.com/sites/default/files/what-is-docker-diagram.png`

Container enthalten die Applikation inklusive aller Abhängigkeiten, teilen sich allerdings den Kernel mit anderen Containern, was zu einem deutlichen geringeren Ressourcenverbrauch führt.
Sofern möglich, werden die einzelnen Teile der Applikation als Docker Container zur Verfügung gestellt.

1.1.2 Dockerfiles

Zur automatisierten Erstellung von Docker Images kann ein Dockerfile bereitgestellt werden. Ein Dockerfile ist ein Text Dokument, das alle Kommandos enthält, die zur Erstellung des images notwendig sind.
Listing 1 zeigt ein beispielhaftes Dockerfile[3]

Listing 1: Dockerfile

```
1 FROM ubuntu:14.04
2 MAINTAINER Kate Smith <ksmith@example.com>
3 RUN apt-get update && apt-get install -y ruby ruby-dev
4 RUN gem install sinatra
```

Im ersten Schritt wird angegeben, welches Basisimage erweitert werden soll. Die Zeile MAINTAINER gibt an wer für die Verwaltung zuständig ist. Im nächsten Schritt wird mittels RUN das Programm apt-get verwendet um die Paketlisten zu aktualisieren und Ruby zu installieren. Als letztes wird ebenfalls mit dem RUN Befehl gem verwendet um Sinatra zu installieren.
Mittel dem Befehl docker build kann das Image erstellt werden. Dockerfiles bieten eine einfache Möglichkeit portable und nachvollziehbare images zu erstellen.
Eine vollständige Beschreibung inklusive aller Kommandos ist unter [2] zu finden.

1.1.3 Compose

Compose ist ein Werkzeug um Anwendung zu definieren und zu starten, die aus mehreren Docker Container bestehen. Dazu wird eine Datei im .yml Format benötigt Listing 2 zeigt ein beispielhaftes docker-compose File[4], das eine Anwendung definiert, die aus zwei Containern besteht.

[2] https://www.docker.com/sites/default/files/what-is-vm-diagram.png
[3] https://docs.docker.com/engine/userguide/dockerimages/
[4] https://docs.docker.com/compose/

Listing 2: Compose

```
web:
  build: .
  ports:
   - "5000:5000"
  volumes:
   - .:/code
  links:
   - redis
redis:
  image: redis
```

Eine vollständige Befehlsreferenz ist unter [3] zu finden.

1.2 Boot2docker

boot2docker ist eine leichtgewichtige Linux Distribution, basierend auf Tiny Core Linux. Sie wurde speziell für den Betrieb von Docker Container entwickelt[4]. Boot2docker lässt sich komplett im RAM starten und verbraucht ca. 27 MB. Ursprünglich diente boot2docker dazu, Docker unter Windows und MacOS zu benutzen, da beide Docker noch nicht nativ unterstützen. Seit 03.05.2015 ist boot2docker deprecated und wurde durch das Tool docker-machine ersetzt [5].

1.3 Rancher OS

Rancher OS ist eine weitere leichtgewichtige Linux Distribution, speziell für den Einsatz von den Docker. Rancher OS besteht komplett aus Docker Containern. Abbildung 3 zeigt den schematischen Aufbau von RancherOS.

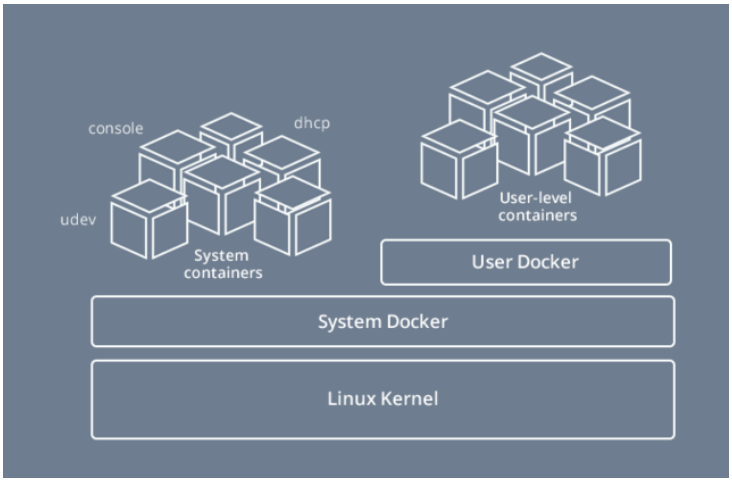


Abbildung 3: Architektur von RancherOS [5]

Beim starten des Systems, werden zwei Instanzen von Docker gestartet. Die erste Instanz (PID1) dient dabei als Ersatz eines klassischen Init Systems. Die zweite Instanz startet und verwaltet die Docker Container des Benutzers. Die beiden Docker Instanzen laufen nebeneinander, das heißt, Änderungen an der einen Instanz haben keine Auswirkungen auf die anderen. RancherOS befindet sich aktuell in einer frühen Entwicklungsphase, mit häufigen, inkompatiblen Änderungen. Dadurch ist RancherOS nicht für den Produktiveinsatz geeignet[6].

1.4 Redis

Redis ist ein Open Source (BSD) lizenzierter, in-memory Datenstruktur Speicher, der sich als Cache, Datenbank und Message Broker benutzen lässt. [7] Zu den unterstützten Datenstrukturen gehören unter anderem:

- Strings
- Hashes
- Listen
- Sets

Redis wird sowohl als zentrale Datenbank der ermittelten Daten, als auch als Zwischenspeicher innerhalb des ELK Stack verwendet.

1.5 Tomcat

Apache Tomcat ist ein weit verbreiteter Open Source (Apache License) Webserver und Java Applikationsserver, der es ermöglicht Anwendungen auf Servlet Basis auszuführen. [8] Die verwendete Instanz ist wie unter Abschnitt 4 beschrieben extra abgehärtet und sicher konfiguriert.

1.6 Alpine Linux

Alpine Linux ist eine Security-orientierte, leichtgewichtige Linux Distribution, die sich Aufgrund ihrer geringen Größe (ca. 5MB) als Basis Image für Docker-Images großer Beliebtheit erfreut. [9]

1.7 Logstash

Logstash ist ein Open Source Programm zur Normalisierung und Zentralisierung von Logdaten aus verschiedensten Quellen[10].
Logstash lässt sich über eine relativ einfache Domain Specific Language (DSL) in JRuby konfigurieren. Dabei gibt man Logstash an, aus welcher Quelle es Daten beziehen soll. Im einfachsten Fall sind das Dateien, möglich sind aber auch IRC-Channel, Twitter-Feeds oder REST-Schnittstellen. Zusätzlich muss angegeben werden, wohin diese Daten geschrieben werden sollen. Meistens ist das für Logstash Elasticsearch, möglich sind hier aber auch Dateien, Emails, diverse Chat-Clients, andere Datenbanken oder Webdienste. Um Daten aufzubereiten oder mit zusätzlichen Informationen anzureichern, kann man Filter festlegen, welche auf Datensätze mit bestimmten Mustern angewendet werden.

1.8 Elasticsearch

Elasticsearch ist eine Open Source Volltext-Suchmaschine, welche auf dem Apache Projekt Lucene basiert[10].
Elasticsearch speichert in einer Schema-losen, Volltext-indizierten in-Memory NoSQL-Datenbank. Dadurch ist es möglich, ohne Änderungen an einem Datenbankschema, Logdaten mit neuen oder geänderten Formaten zu speichern. Nach einem Update der Indizes können diese neuen Formate auch in Volltext-Suchen verwendet werden.

1.9 Kibana

Kibana ist ein Open Source Plugin und Web-Frontend für Elasticsearch um Daten zu visualisieren[10]. Dadurch lassen sich Ausgaben von Elasticsearch-Queries in Diagrammen oder auf Zeitachsen plotten.

1.10 Shield

Ähnlich wie Elasticsearch, Logstash und Kibana wird Shield von Elastic entwickelt, im Gegensatz zu den vorher genannten Produkten ist Shield allerdings nicht Open Source. Shield ist eine Zusatzsoftware für Elastic-Produkte um Authentifizierung, Autorisierung und Verschlüsselung zu ermöglichen, da dies standardmäßig nicht möglich ist[10].

1.11 stunnel

stunnel ist eine Open Source Software, mit dem eine bestehende Client-Server-Architektur nachträglich mit TLS-Verschlüsselung und Signierung ausgestattet werden kann[11].

1.12 nginx

Nginx ist ein sehr leichtgewichtiger Open Source Webserver, der sich besonders für die Benutzung als Reverse Proxy eignet[12]. In diesem Projekt werden wir ihn als Reverse Proxy für Kibana verwenden.

1.13 eCryptfs

eCrytptfs ist ein kryptographisches Dateisystem für Linux[13].

2 Spezifikation

2.1 Funktionalität

Zentrale Überwachung (Monitoring) der Systeme und Dienste in einem Heimnetzwerk und Visualisierung dieser. Der Fokus innerhalb der einzelnen Komponenten wie z.B. Datenbank und Webserver liegt auf der Härtung und Isolation mittels Docker.

2.2 Existierende Lösungen zur Systemüberwachung

Es gibt bereits einige Softwarelösungen, um ein zentrales Monitoringsystem zu realisieren. Im folgenden werden die Produkte „Nagios“ und „Splunk“ genauer betrachtet.

- **Nagios:** Das Unternehmen Nagios beschäftigt sich mit der Realisierung einer Überwachung von IT-Infrastrukturen. Es können Systemmetriken, Netzwerkprotokolle, Anwendungen, Dienste, Server, und die Netzwerkinfrastruktur überwacht werden.

- **Splunk:** Bei Splunk handelt es ebenfalls sich um ein großes Unternehmen, dass sich darauf spezialisiert hat mit ihrer Software beliebige Maschinendaten mehrerer Systeme überwachen zu können. Das nachfolgenden Bild zeigt die verschiedenen Daten, die mit Splunk überwacht werden können.

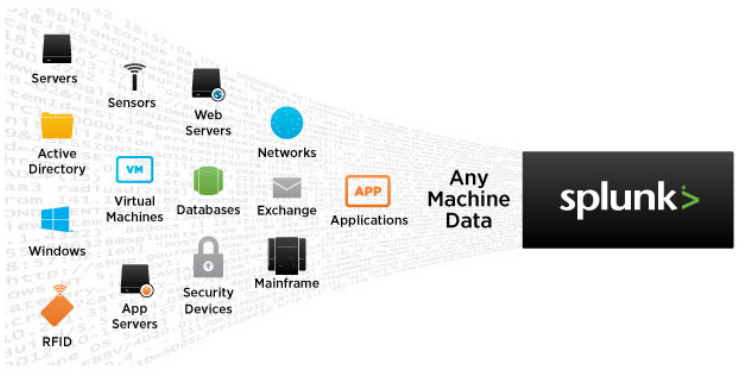


Abbildung 4: Splunk Architektur [(6)]

2.3 Detailfunktionalitäten

- Benutzung einer speziellen Linux Distribution für den Einsatz von Docker

[(6)] http://www.splunk.com/en_us/resources/operational-intelligence.html

  - boot2docker[7]
  - RancherOS[8]

- Monitoring Webapplikation: Beim Monitoring wird unterschieden zwischen
  - lokalem Monitoring:
    * SSH: Zugriffe, Anmeldeversuche, etc...
    * Docker Container: Welche Container laufen gerade, welche sind vorhanden, Starten/Stoppen, etc...
    * Systemparameter: Freier Festplattenspeicher, laufende Prozesse, RAM, CPU, etc...
  - Remote Monitoring (andere Gruppen):
    * Logauswertung
    * Applikationsüberwachung
    * IDS Überwachung
    * etc...

- Authentifizierung innerhalb der Webapplikation gegenüber GitHub mittels OAuth
  - Speichern der erfolgreichen und fehlgeschlagenen Authentifizierungen innerhalb einer Datenbank und Visualisierung dieser
  - Autorisierung über GitHub-Teams
    * Mitglieder des SichHeimMonitor GitHub Teams dürfen alles (wobei alles noch zu definieren ist)
    * GitHub Mitglieder haben eingeschränkten Zugriff (wobei eingeschränkt noch zu definieren ist)
    * Anonyme Benutzer haben keinen Zugriff

- Konfiguration und Härtung des ELK Stack (Elasticsearch, Logstash & Kibana)
  - Der ELK Stack ist eine echtzeitfähige Lösung zur Analyse verschiedener Arten von strukturierten und unstrukturierten Daten.

[7] `http://boot2docker.io/`
[8] `http://rancher.com/rancher-os/`

  - Sammeln und Aufbereiten der Logdaten lokaler und externer Prozesse mit Logstash
  - Speicherung der Daten in Elasticsearch (evtl. anbieten einer API)
  - Visuelle Darstellung der Logdaten mit Kibana (Monitoring Webapplikation)

2.4 Sicherheit

- Authentifizierung mittels OAuth
- Autorisierung durch GitHub-Team
- Härtung durch Virtualisierung/Isolation
- Isolation der einzelnen Komponenten
- Sichere Kommunikation zur Webapplikation
- Sichere Speicherung der Daten

2.5 Schnittstellen

- Logstash-Shipper (Sammeln Logdaten der anderen Gruppen)
- Remotedaemon (Falls Gruppen kein konventionelles Logging verwenden, Erfassen der Systemparameter)

2.6 Architektur-/Realisierungsübersicht als Grafik

In der folgenden Grafik ist die Gesamtarchitektur des sicheren Heimmonitoring Systems dargestellt. Die Blauen Vierecke stehen hierbei für einzelne und isolierte Docker Images.

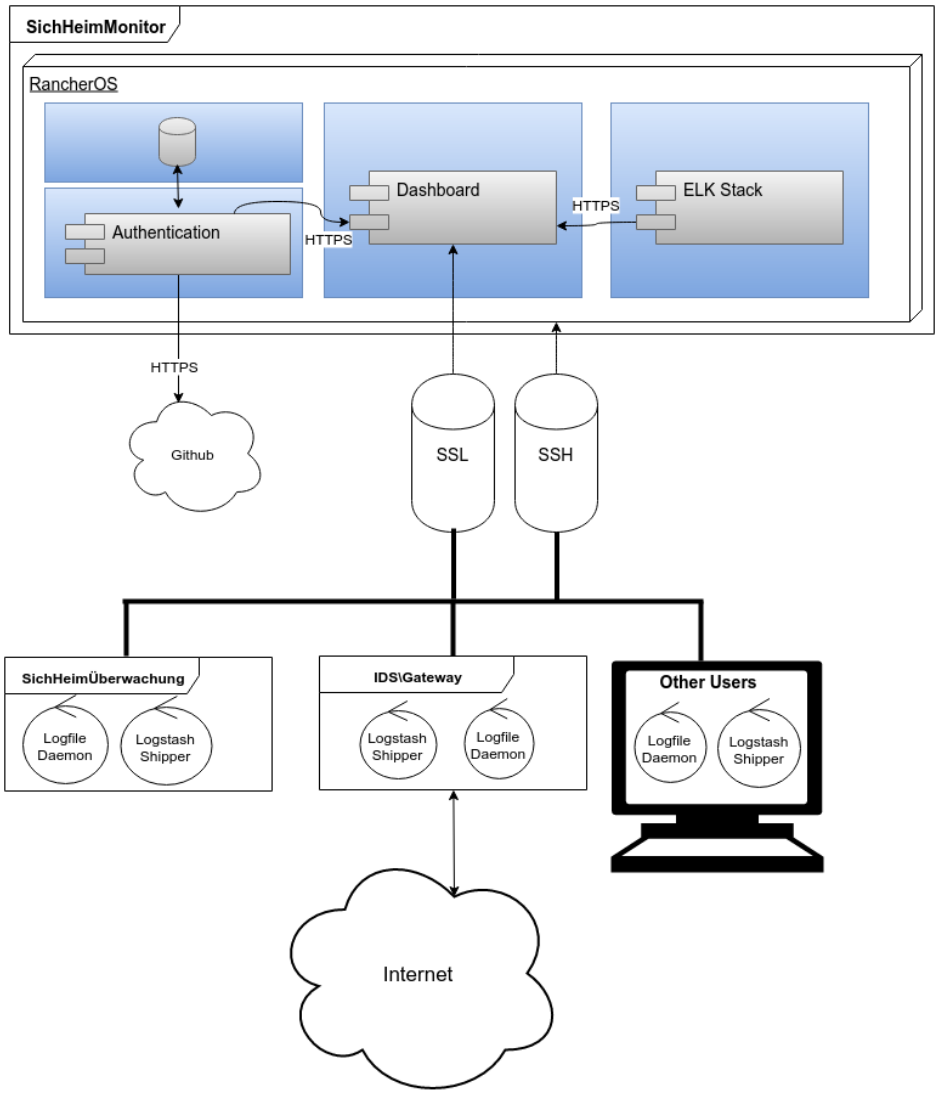


Abbildung 5: Sicheres Heimmonitoring Architektur

2.7 Schutzbedarfsfestellung

2.7.1 Funktionalität

Für die Schutzbedarfsfeststellung liegen folgende Anwendungsfälle zugrunde

Anwendungsfall	Beschreibung
UC1	Der Benutzer hat die Möglichkeit, eine grafische Logauswertung (andere Dienste/Systeme) durchzuführen.
UC2	In der Webapp können die einzelnen Dienste gesteuert werden.
UC3	Es gibt die Möglichkeit einzelne Dienste in der Webapp aufzulisten.
UC4	Der Überwachungsserver kann über die Webapp verwaltet werden.
UC5	Ein Benutzer kann einen anonymisierten Browser über die Webapp starten.

2.7.2 Sicherheitsanforderungen

In diesem Abschnitt werden die Sicherheitsanforderungen definiert und gewichtet. Bei den Gewichtungen wird unterschieden zwischen Gering, Mittel und Hoch. Die folgenden Sicherheitsanforderungen werden betrachtet:

- Vertraulichkeit (Confidentiality): Die Informationen, Anwendungen, Dienste usw. dürfen nur Berechtigten zugänglich sein.
  - Gewichtung: Hoch
  - Begründung: Im Monitoringsystem werden teilweise vertrauliche/-personenbezogene Daten verarbeitet.
  - Beispiel: Gespeicherte Logdaten aus der Heimüberwachung.
- Integrität (Integrity): Die Informationen, Anwendungen, Dienste usw. dürfen nur von Berechtigten verändert werden.
  - Gewichtung: Hoch
  - Begründung: Durch unerlaubtes Bearbeiten von Informationen kann dem Benutzer ein falsches Gefühl von Sicherheit vermittelt werden.
  - Beispiel: Die Heimüberwachung stellt einen Einbruch fest, die Hinweise darauf werden vom Einbrecher aus den Logdaten entfernt.
- Verfügbarkeit (Availability): Die Informationen, Anwendungen, Dienste usw. sind in einer zugesicherten Zeit zugänglich.

  - Gewichtung: Mittel
  - Begründung: Die angebotenen Funktionen dienen lediglich der Bequemlichkeit.

  Die überwachten Anwendungen sind davon nicht betroffen.

- Authentizität (Authenticity): Die Echtheit/Urheberschaft einer Information, Anwendung, Dienst usw. ist nachweisbar.

  - Gewichtung: Hoch
  - Begründung: Logdaten können kritische Informationen enthalten, z.B. den aktuellen

  Systemstatus. Wird dieser von aussenstehenden manipuliert können gezielt falsche Infomrationen für Angriffe benutzt werden.

- Datenschutz (Privacy): Der Schutz personenbezogener Daten vor Mißbrauch und der Schutz der Privatsphäre.

  - Gewichtung: Hoch
  - Begründung: Im Monitoringsystem werden teilweise vertrauliche/-personenbezogene Daten verarbeitet.
  - Beispiel: Gespeicherte Logdaten aus der Heimüberwachung.

2.7.3 Gefahren

Für die angebotenen Funktionalitäten haben wir im Hinblick auf den Gefährdungskatalog der Elementaren Gefährdungen[14] des IT-Grundschutz-Kataloges folgende relevante Gefahren ermittelt:

Gefahr	Beschreibung	Referenz in BSI	Wirkt auf Anforderung
G01	Ausfall der Stromversorgung oder des Kommunikationsnetze	G 0.8ff	Verfügbarkeit
G02	Ausspähung oder Abhören von Informationen	G 0.14f	Vertraulichkeit, Datenschutz
G03	Verlust/Diebstahl von Geräten, Datenträgern oder Dokumenten	G 0.16f, G 0.19, G 0.45	Vertraulichkeit, Verfügbarkeit, Datenschutz
G04	Manipulation von Hard- oder Software	G 0.21	Authentizität, Integrität
G05	Manipulation von Informationen, Einspielen von Nachrichten	G 0.22, G 0.43, G 0.46	Authentizität, Integrität
G06	Unbefugtes Eindringen in IT-Systeme	G 0.23	Vertraulichkeit, Datenschutz
G07	Zerstörung von Geräten oder Datenträgern, Sabotage	G 0.24, G 0.41	Verfügbarkeit
G08	Ausfall/Fehlfunktion von Geräten oder Systemen	G 0.25f	Verfügbarkeit
G09	Software-Schwachstellen oder -Fehler, Schadprogramme	G 0.28, G 0.39	alle genannten
G10	Unberechtigte Nutzung oder Administration von Geräten und Systemen	G 0.30	alle genannten
G11	Missbrauch personenbezogener Daten	G 0.38	Vertraulichkeit, Datenschutz
G12	Unbefugtes Eindringen in Räumlichkeiten	G 0.44	Vertraulichkeit, Datenschutz

Nicht näher betrachtet werden dabei Gefahren, bei denen der Schaden am Monitoring-System gegenüber anderweitig entstehender Schäden nicht wirklich relevant ist, beispielsweise Feuer oder Naturkatastrophen.

2.7.4 Risiken

Um das Risiko einzuschätzen, wird folgenden Risikotabelle verwendet:

Schadenspotential/ Eintrittswahrscheinlichkeit	**selten**	**manchmal**	**oft**	**sehr oft**
existenzbedrohend ($>$ 50.000 €)	**hoch**	**hoch**	**hoch**	**hoch**
hoch (1-50k €)	mittel	mittel	**hoch**	**hoch**
mittel (bis 1000 €)	gering	gering	mittel	**hoch**
gering	gering	gering	gering	mittel

Durch Betrachtung der Eintrittswahrscheinlichkeit und des Schadenspotentials der einzelnen Gefahren, wird ermittelt, welche Risiko diese darstellt.

1. G01: Ausfall der Stromversorgung oder des Kommunikationsnetze
    - Schaden: gering
    - Eintrittswahrscheinlichkeit: manchmal
    - Resultierendes Risiko: gering
    - Begründung: Stromversorgung und Internetverbindung sollte stabil sein (Monitoring ist bei Ausfall kleinste Sorge)
2. G02: Ausspähung von Informationen oder Abhören
    - Schaden: mittel
    - Eintrittswahrscheinlichkeit: manchmal
    - Resultierendes Risiko: gering
    - Begründung: Monitoring findet komplett im internen Netzwerk statt. Gefahr nur durch Gäste oder Einbrecher.
3. G03: Verlust/Diebstahl von Geräten, Datenträgern oder Dokumenten
    - Schaden: hoch
    - Eintrittswahrscheinlichkeit: selten
    - Resultierendes Risiko: **mittel**
    - Begründung: Gefahr nur durch Gäste oder Einbrecher.
4. G04: Manipulation von Hard- oder Software
    - Schaden: mittel

- Eintrittswahrscheinlichkeit: manchmal
- Resultierendes Risiko: gering
- Begründung: Gefahr nur durch Gäste oder Einbrecher.

5. G05: Manipulation von Informationen, Einspielen von Nachrichten
    - Schaden: mittel
    - Eintrittswahrscheinlichkeit: manchmal
    - Resultierendes Risiko: gering
    - Begründung: Monitoring findet komplett im internen Netzwerk statt. Gefahr nur durch Gäste oder Einbrecher.
6. G06: Unbefugtes Eindringen in IT-Systeme
    - Schaden: mittel
    - Eintrittswahrscheinlichkeit: selten
    - Resultierendes Risiko: gering
    - Begründung: Gefahr nur durch Gäste oder Einbrecher.
7. G07: Zerstörung von Geräten oder Datenträgern, Sabotage
    - Schaden: hoch
    - Eintrittswahrscheinlichkeit: selten
    - Resultierendes Risiko: gering
    - Begründung: Gefahr nur durch Gäste oder Einbrecher.
8. G08: Ausfall/Fehlfunktion von Geräten oder Systemen
    - Schaden: gering
    - Eintrittswahrscheinlichkeit: manchmal
    - Resultierendes Risiko: gering
    - Begründung: Verfügbarkeit des Monitoring-Systems wird als weniger wichtig angesehen.
9. G09: Software-Schwachstellen oder -Fehler, Schadprogramme
    - Schaden: mittel

- Eintrittswahrscheinlichkeit: oft
- Resultierendes Risiko: **mittel**
- Begründung: Gefahr nicht auszuschließen, erstreckt sich über alle Anforderungen.

10. G10: Unberechtigte Nutzung oder Administration von Geräten und Systemen
    - Schaden: mittel
    - Eintrittswahrscheinlichkeit: sehr oft
    - Resultierendes Risiko: **hoch**
    - Begründung: Ohne Authentifizierung von jedem Teilnehmer im Netz ausnutzbar.
11. G11: Missbrauch personenbezogener Daten
    - Schaden: mittel
    - Eintrittswahrscheinlichkeit: oft
    - Resultierendes Risiko: **mittel**
    - Begründung: Gefahr nur durch Gäste oder Einbrecher.
12. G12: Unbefugtes Eindringen in Räumlichkeiten
    - Schaden: gering
    - Eintrittswahrscheinlichkeit: selten
    - Resultierendes Risiko: gering
    - Begründung: Gefahr nur durch Einbrecher. Interesse der Einbrecher sicher nicht das Monitoring-System.

2.7.5 Sicherheitskonzeption

Um die größten Risiken zu beseitigen werden diese Maßnahmen implementiert:

Maßnahme	Beschreibung
M01	Verschlüsselte Datenübertragung
M02	Verschlüsselte Speicherung der Daten
M03	Signierte Datenübertragung
M04	Benutzerauthentifizierung über OAuth
M05	Replizierung wichtiger Dienste
M06	Härtung und Konfiguration der einzelnen Komponenten

Zu Maßnahme M04 ist anzumerken, dass die Authentifizierung über das Internet stattfindet, sie muss deshalb entsprechend gesichert werden.
Dadurch ergibt sich die folgende Abdeckung der Risiken:

Gefahr	Risiko	Maßnahme
G01	gering	-
G02	gering	M01
G03	**mittel**	M02
G04	gering	-
G05	gering	M03
G06	gering	M04
G07	gering	-
G08	gering	M05
G09	**mittel**	M06
G10	**hoch**	M04
G11	**mittel**	M01
G12	gering	-

3 Überwachungsdienst

3.1 Einleitung

Damit eine zentrale Überwachung(Monitoring) der Systeme in einem Heimnetzwerk realisiert werden kann, muss es eine Komponente geben, die die zu überwachenden Informationen der einzelnen Systeme ermittelt und an das zentrale Überwachungssystem überträgt. Diese Aufgabe wird von einem Dienst übernommen. Dieser Dienst wird auf den einzelnen Systemen installiert und bedarfsabhängig konfiguriert. Zu der bereits formulierten Anwendungsfällen in Abschnitt Funktionalität der Spezifikation, die sich auf die Gesamtanwendung bezogen haben, kommen für den Überwachungsdienst noch folgende funktionale Anforderungen hinzu:

Tabelle 1: Funktionale Anforderungen für den Überwachungsdienst

Funktionale Anforderung	**Beschreibung**
FA01	Der Dienst kann neue Einträge in Logdateien erkennen.
FA02	Der Dienst kann regelmäßig einen Befehle ausführen und die Standardausgabe ermitteln.
FA03	Der Benutzer soll die Möglichkeit haben die Überwachung konfigurieren zu können.
FA04	Der Benutzer kann Einträge, die von der Logdateiüberwachung erkannt werden, filtern.
FA05	Der Benutzer hat die Möglichkeit für die Logdateiüberwachung verschiedene Kritikalitäten zu definieren.
FA06	Der Dienst soll die gesammelten Daten gesichert an das zentrale Überwachungssystem übermitteln.

3.2 Architektur

Im folgenden wird die Architektur des Überwachungsdienstes beschrieben. Zunächst wird aufgezeigt, welche Aufgaben der Überwachungsdienst in der Gesamtarchitektur übernimmt. Anschließend wird auf den Aufbau des Überwachungsdienstes eingegangen.

3.2.1 Einordnung in die Gesamtarchitektur

Der Überwachungsdienst muss zunächst auf die Systeme installiert werden. Anschließend ist die Überwachung des Systems aktiv. Abbildung 6 zeigt einen Ausschnitt der Gesamtarchitektur. Der Dienst ist in der Grafik in rot dargestellt und sendet die erhobenen Daten über eine sichere Verbindung (SSL) an das Überwachungssystem (SichHeimMonitor). Die Komponente, die dabei die Daten empfängt und in eine Datenbank ablegt ist blau dargestellt. Das bedeutet das es eine **zentrale Datenhaltung** gibt.

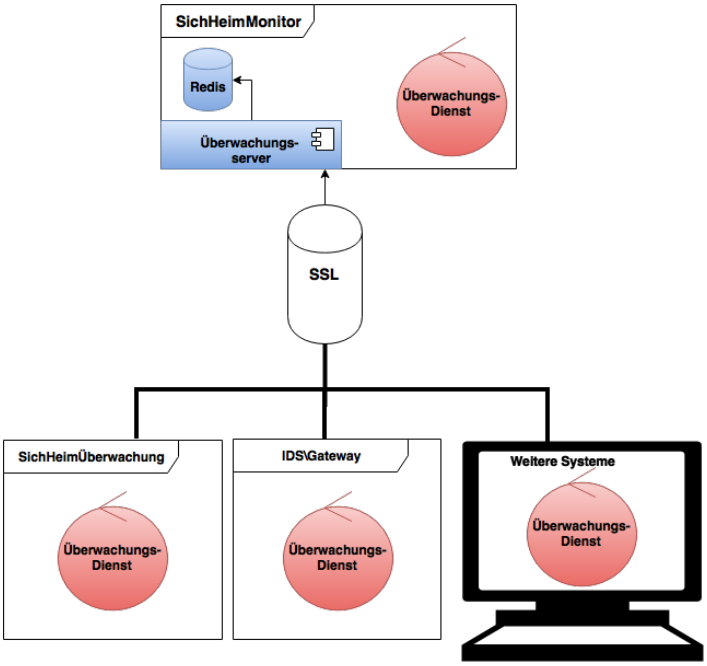


Abbildung 6: Einordnung in die Gesamtarchitektur

3.2.2 Architektur des Überwachungsdienstes

Der Überwachungsdienst besteht aus sieben Klassen und zwei Dokumenten. Abbildung 7 zeigt auf, in welchen Beziehungen die einzelnen Klassen und Dokumente zueinander stehen. Anschließend werden kurz die Aufgaben der einzelnen Klassen und Dokumente erläutert. Die blau Dargestellten Komponenten sind Hilfsklassen und sorgen dafür, dass die Daten, die bei der Überwachung anfallen formatiert und an den zentralen Überwachungsserver

gesendet werden. Die grünen Bereiche kümmern sich um die Konfiguration und den Start des Dienstes, während die in rot dargestellten Teile die eigentliche Überwachung steuern. Das Client-Zertifikat, welches vom Dienst verwendet wird, ist hier nicht dargestellt.

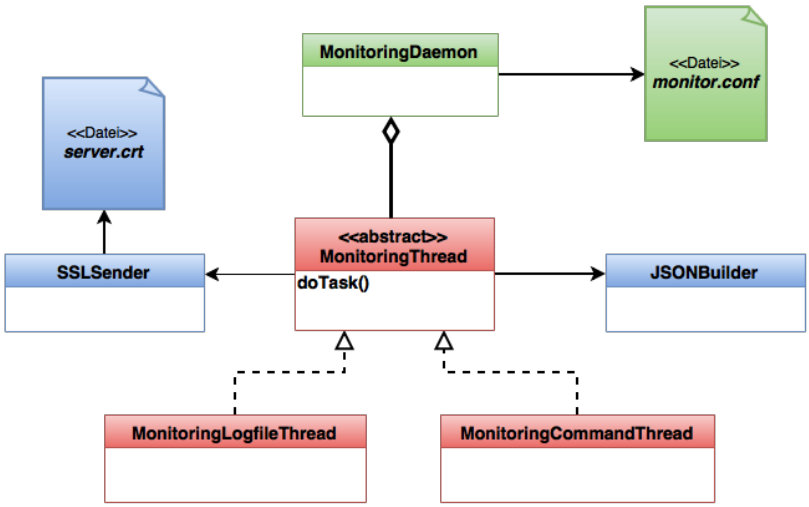


Abbildung 7: Architektur des Überwachungsdienstes

- **server.crt:** In der Datei `server.crt` befindet sich ein Zertifikat mit einem öffentlichen 2048 Bit RSA Schlüssel. Damit wird eine Verschlüsselung über SSL mit dem zentralen Überwachungsserver ermöglicht. Zusätzlich wird ein Client-Zertifikat benötigt. Dieses ist in der Grafik nicht abgebildet.

- **monitor.conf:** In dieser Datei kann die Konfiguration der Anwendung vorgenommen werden. Es kann eingestellt werden, wie der zentrale Überwachungsserver erreicht werden kann. Außerdem kann bestimmt werden, welche Daten des Systems überwacht werden sollen.

- **MonitoringDaemon:** Die Klasse `MonitoringDaemon` ist ein Thread und gleichzeitig Haupteinstiegspunkt der Anwendung. Hier wird die Datei `monitor.conf` verwendet, um mehrere Threads der Klasse `MonitoringThreads` zu starten. Das Programm beendet sich erst, wenn ein Interruptsignal an den Prozess gesendet wird.

- **MonitoringThread:** Der `MonitoringThread` ist eine abstrakte Basisklasse und bietet ihren Unterklassen die Möglichkeit verschiedene Überwachungsaufgaben in einem bestimmten Intervall ausführen zu können. Nach jedem Intervall wird die Methode `doTask()` aufgerufen. Diese Methode erhebt die zu überwachenden Daten und verwendet die Klassen `SSLSender` und `JSONBuilder`, um diese an den zentralen Überwachungsserver zu senden.

- **MonitoringLogfileThread:** Mit der Klasse `MonitoringLogfileThread` können Logdateien überwacht werden.

- **MonitoringCommandThread:** Mit der Klasse `MonitoringCommandThread` kann die Ausgabe bestimmter Befehle überwacht werden. Das Kommando `df -h | sed '1d' | awk {'print $1, $5'}` könnte zum Beispiel verwendet werden, um den freien Speicherplatz eines Systems zu ermitteln.

- **SSLSender:** Die Klasse `SSLSender` übernimmt die Aufgabe die Daten zu kodieren und über SSL an den zentralen Monitoringserver zu übermitteln.

- **JSONBuilder:** Mit der Hilfsklasse `JSONBuilder` werden die erhobenen Daten in einen JSON-String konvertiert.

3.2.3 Architektur des Überwachungsservers

Die von dem Dienst erhobenen Daten werden an einen Überwachungsserver gesendet. Der Server überprüft die Daten und speichert diese in einer Datenbank ab. In Abbildung 8 wird die Architektur des Überwachungsservers dargestellt.

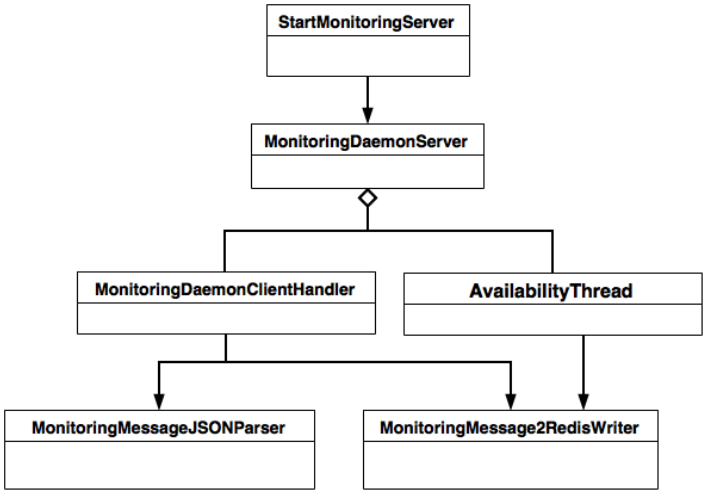


Abbildung 8: Architektur des Überwachungsservers

- **StartMonitoringServer:** Die Klasse `StartMonitoringServer` startet den Thread `MonitoringDaemonServer`.
- **MonitoringDaemonServer:** Diese Klasse öffnet einen SSLServer-Socket und startet für jede eingehende Verbindung einen Thread des Typs `MonitoringDaemonClientHandler`. Außerdem wird mit der Verfügbarkeitsüberwachung, durch starten mehrerer Threads der Klasse `AvailabilityThread`, begonnen.
- **MonitoringDaemonClientHandler:** Die Aufgabe dieser Klasse ist das Empfangen der eingehenden Überwachungsdaten. Darüberhinaus überprüft diese Klasse die Daten und schreibt diese in eine Datenbank. Dabei werden die Klassen `MonitoringMessageJSONParser` und `MonitoringMessage2RedisWriter` verwendet. Es wird überprüft, ob der Client noch aktiv ist. Dadurch kann festgestellt werden, ob die Überwachung noch aktiv ist.
- **AvailabilityThread:** Diese Klasse verwendet das Kommandozeilenprogramm `ping`, um in regelmäßigen Abständen, die Verfügbarkeit der Systeme zu überprüfen.
- **MonitoringMessageJSONParser:** Diese Hilfsklasse gibt Felder eines JSON-Strings zurück.

- **MonitoringMessage2RedisWriter:** Diese Hilfsklasse nimmt eine Verbindung zu einer Redis Datenbank auf und schreibt eintreffenden Daten in die Datenbank.

3.3 Implementierung

In diesem Abschnitt wird auf die Implementierung des Überwachungsdienstes eingegangen. Der Dienst wurde in der Programmiersprache Python realisiert. Der Überwachungsserver ist in Java programmiert. Zunächst wird der allgemeine Programmablauf des Diestes beschrieben. Anschließend wird der Aufbau der Konfigurationsdatei dargelegt.

1. Der Dienst sammelt die zu überwachenden Daten.
2. Der Dienst sendet Daten an den zentralen Überwachungsserver.
3. Der zentrale Überwachungsserver legt die empfangenen Daten in Redis ab.
4. Das Dashboard entnimmt aktuelle Überwachungsdaten aus Redis.

3.3.1 Allgemeiner Programmablauf des Dienstes

Der Dienst muss beim hochfahren des Systems gestartet werden. Die Möglichkeiten dies zu realisieren wird in Kapitel Installation und Konfiguration des Dienstes beschrieben. Anschließend verwendet er die in der Konfigurationsdatei `monitor.conf` angegebene IP-Adresse und den Port um eine Verbindung mit dem zentralen Überwachungsservers aufzubauen. Nach einem erfolgreichen Verbindungsaufbau, wird für jede weitere syntaktisch korrekte Konfigurationszeile der Konfigurationsdatei ein Thread gestartet. Die Threads beginnen daraufhin mit der Überwachung des Systems. Wenn das System heruntergefahren wird und somit ein Interruptsignal an den Dienst gesendet wird, beendet der Dienst alle gestarteten Threads und endet anschließend selbst. In Abbildung 9 wird der Programmablauf anhand eines Ablaufdiagramms dargestellt.

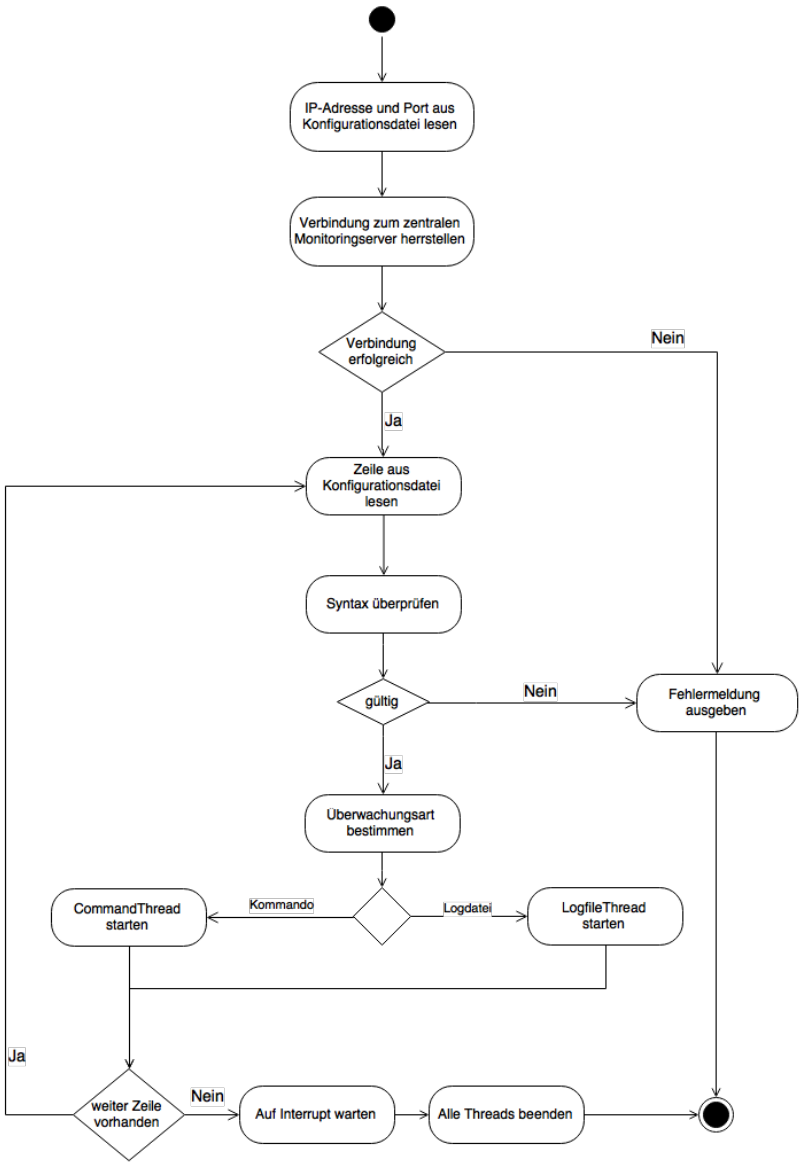


Abbildung 9: Daemon Ablaufdiagramm

3.3.2 Aufbau der Konfigurationsdatei

Der Dienst verwendet eine Konfigurationsdatei mit dem Namen `monitor.conf`. Durch Einführung einer Konfigurationsdatei hat der An-

wender die Möglichkeit, zu bestimmen welche Daten des Systems von dem Dienst überwacht werden sollen. Das hat den Vorteil, dass der Dienst für verschiedene Systeme individuell konfiguriert werden kann. Im folgenden wird der syntaktische Aufbau der Konfigurationsdatei erläutert.

In der Konfigurationsdatei werden Leerzeilen und Zeilen die mit einem Doppelkreuz(#) beginnen ignoriert. Somit besteht die Möglichkeit mit dem Doppelkreuz die Datei mit Zeilenkommentaren zu versehen. In der Datei müssen die Schlüsselwörter „IP“ und „Port“ spezifiziert werden. Diese Angaben werden benötigt, damit der Dienst eine Verbindung zum zentralen Überwachungsserver aufnehmen kann. Durch folgende Syntax in der Datei können die Schlüsselwörter einem Wert zugeordnet werden:
IP: {IP-Nummer des Überwachungsservers}
Port: {Port-Nummer des Überwachungsservers}

Die anderen Konfigurationszeilen, die nicht leer sind und nicht mit dem Doppelkreuz oder einem Schlüsselwort beginnen, werden zur Steuerung der Systemüberwachung herangezogen. Diese Zeilen müssen eine der beiden Syntaxen entsprechen, die in der folgenden Tabelle angegeben sind:

Tabelle 2: Mögliche der Konfigurationszeilen

	Syntax
Logdatei	`c#command#topic#intervall_in_ms#\ [criticality#regex_for_logfile(optional)]`
Kommando	`l#logfile_path#topic#intervall_in_ms`

Zur Konfiguration einer Logdateiüberwachung wird die Syntax, die in der ersten Spalte angegeben ist benötigt. Wenn die Ausgabe eines Befehls überwacht werden soll, muss die Konfiguration mit der Syntax der zweiten Spalte vorgenommen werden. Beide Konfigurationszeilen verwenden als Trennzeichen zwischen ihren Werten ein Doppelkreuz und müssen mindestens vier Werte enthalten. Im folgenden werden die einzelnen Felder und die dazugehörigen Werte eingeführt:

Pflichtfelder: Diese Felder müssen sowohl bei Logdateien als auch bei Kommandos angegeben werden.

- **Feld 1:** Dieses Feld darf nur die Werte `c` oder `l` annehmen. Wenn der

Wert `c` angegeben wird, wird dem Dienst bekannt gemacht, dass nun eine Konfigurationszeile für ein Kommando folgt. Durch Angabe des Werts `l` geht der Dienst davon aus, dass eine Konfigurationszeile für eine Logdatei folgt.

- **Feld 2:** In diesem Feld wird eine Zeichenkette erwartet. Abhängig vom ersten Feld wird die Zeichenkette entweder als Kommando oder als Dateipfad interpretiert. Bei Angabe dieses Feldes sollte vorher überprüft werden, ob der Betriebssystembenutzer, der den Dienst startet, die erforderlichen Rechte besitzt, um auf die angegebene Logdatei zugreifen bzw. den angegebenen Befehl ausführen zu können.

- **Feld 3:** Mit diesem Feld kann bestimmt werden, wie die erhobenen Daten auf dem Dashboard dargestellt werden. Darüberhinaus kann dieses Feld verwendet werden, um die erhobenen Daten mehrerer Konfigurationszeilen zu gruppieren. Es wird erwartet, dass der Inhalt dieses Feldes mit einem vordefinierten Präfixe beginnt. Konfigurationzeilen, die in diesem Feld die gleiche Zeichenkette verwenden, werden auf dem zentralen Überwachungssystem im selben Kontext gebündelt dargestellt. Im folgenden wird aufgezeigt, wie die einzelnen Präfixe vom zentralen Überwachungssystem interpretiert werden.

  - **ram:** Mit diesem Präfix wird dem zentralen Überwachungssytem mitgeteilt, dass die folgenden Daten als Liniendiagramm dargestellt werden sollen. Damit das zentrale Überwachungssystem die Daten richtig interpretieren kann, müssen diese als zwei Zahlen mit Leerzeichen getrennt übermittelt werden. Die erste Zahl soll dabei den gesamten verfügbaren RAM angeben und die zweite Zahl soll den momentan verwendeten RAM angeben.
  - **cpu:** Die Daten mit diesem Präfix werden ebenfalls als Liniendiagramm angezeigt. Die Auslastung einer CPU wird in einer Zeile angeben. Wenn es mehrere CPUs gibt oder die Auslastung mehrerer Kerne angezeigt werden soll, müssen zusätzliche Zeilen hinzugefügt werden. Eine Zeile muss zwei Werte enthalten. Der erste Wert ist eine beliebige Zeichenkette, die beschreibt, für welchen CPU bzw. Kern die Auslastung übermittelt wird. Der zweite Wert ist eine Zahl, die die Auslastung der CPU bzw. des Kerns in Prozent angibt.

  - **hdd:** Daten, die mit diesem Präfix markiert sind, werden als Tortendiagramm im zentralen Überwachungssytem angezeigt. Die Daten müssen zwei Zahlen enthalten. Die erste Zahl gibt den gesamten Speicherplatz des Systems an, während die zweite Zahl den verwendet Speicherplatz angibt.
  - **command:** Diese Daten werden vom zentralen Überwachungssytem als Text angezeigt. Und können beliebigen Inhalt haben.
  - **logfile:** Diese Daten werden je nach Kritikalität vom zentralen Überwachungssytem als Text in verschiedenen Farben angezeigt. Der Inhalt dieser Daten ist beliebig.

- **Feld 4:** In diesem Feld muss eine Zahl eingegeben werden. Diese Zahl legt ein Zeitintervall in Millisekunden fest, in dem der Dienst die zu überwachenden Daten erhebt. Bei einer Logdateiüberwachung bedeutet das, dass die Logdatei von der zuletzt eingelesenen Zeile anfängt, die Datei zu scannen. Bei der Überwachung von der Ausgabe eines Kommandos wird nach Ablauf des Zeitintervalls das Kommando ausgeführt.

Optionale Felder: Diese beide optionalen Felder können nur für eine Logdateiüberwachung angegeben werden.

- **Feld 5:** Diese optionale Angabe legt eine Kritikalität für die Überwachung eine Logdatei fest. Es gibt die folgenden drei verschiedene Stufen, die hier angegeben werden können:

  - **INFO:** Die erhobenen Daten werden an das zentrale Überwachungssystem gesendet und angezeigt.
  - **WARNING:** Wenn Meldungen für diese Kritikalität an den zentralen Überwachungssystem eingeht, werden diese in einem anderen Bereich dargestellt.
  - **CRITICAL:** Alle kritische Daten, die auf den zentralen Überwachungssystem eintreffen werden auch in einem anderen Bereich dargestellt. Der Benutzer sieht durch eine markierte Ausgabe auf dem Dashboard sofort, wenn kritische Meldungen eintreffen.

- **Feld 6:** Dieses optionale Feld erwartet einen regulären Außdruck. Der reguläre Außdruck muss die Syntax aufweißen, die von dem Python Modul `re` akzeptiert wird. Die Syntax solcher regulärer Außdrücke kann in

der Python-Dokumentation[9]nachgeschlagen werden. Der Dienst ignoriert alle Zeilen der Logdatei, die nicht dem angegebenen Muster entsprechen.

3.3.3 Logdateiüberwachung

In Logdateien protokollieren Prozesse, welche Aktionen vorgenommen werden oder welche Fehler bei der Programmausführung auftreten. In der Regel ist es nicht möglich vorherzusagen, wann ein Prozess einen Eintrag in eine Logdatei schreibt. Es gibt somit keine effiziente Möglichkeit eine Logdateiüberwachung durchzuführen.

Der Dienst lößt dieses Problem dadurch, dass es in regelmäßigen Abstand die Logdatei auf Änderungen überprüft. Die Verfolgung dieses Lösungswegs führt jedoch dazu, dass der Dienst protokolieren muss, welche Zeile als letztes gelesen wurde. Der Ablauf der Logdateiüberwachung lässt sich in drei Schritten zusammenfassen. Der erste Schritt ist das initale Einlesen der Logdatei. Der zweite Schritt ist das überprüfen der Logdatei. Der dritte Schritt ist das Lesen und ggf. Filtern der neuen Zeilen. Bei jedem weiteren Programmdurchlauf wird nur noch Schritt zwei und drei durchgeführt. Im folgenden werden die drei Schritte genauer erläutert

- **Schritt 1: Initiales Einlesen**
  Wenn ein Thread der Klasse `MonitoringLogfileThread` gestartet wird, wird die zu überwachende Logdatei eingelesen. Der Thread merkt sich die zuletzt gelesene Zeilennummer und die Dateigröße. Dies ist notwendig, damit bereits gelesene Zeilen der Logdatei nach einem Neustart des Dienstes nicht erneut erfasst werden. Nachfolgender Code zeigt die Implementierung des initalen Einlesens der Logdatei:

Listing 3: Initiales ermitteln der Zeilennummern

```
1 self.size = os.stat(file_path).st_size
2
3 with open(self.file_path) as f:
4     self.last_position = sum(1 for line in f)
```

[9] `https://docs.python.org/2/library/re.html`

- **Schritt 2 Logdatei überprüfen**
  In den meisten Linux-Betriebssystemen wird das Werkzeug `logrotate` eingesetzt. Dieses Hilfsprogramm sorgt dafür, dass eine Logdatei nicht beliebig groß wird. Logdateien werden in einem bestimmen Intervall und ab einer bestimmten Dateigröße umbenannt und komprimiert. Für den Überwachungsdienst bedeutet das, dass eine Überprüfung stattfinden muss, ob das Programm `logrotate` ausgeführt wurde. Je nachdem, wie `logrotate` konfiguriert ist, kann es vorkommen, dass die Logdatei nicht mehr existiert oder deutlich weniger Zeilen enthält. Wenn die Logdatei nicht existiert, muss gewartet werden, bis der Prozess der die Protokollierung dort vornimmt, die Datei wieder erstellt. Falls die Datei existiert und eine kleinere Dateigröße hat als beim letzten Durchlauf, wird die vermerkte Zeilennummer wieder auf Null zurückgesetzt. Das Zurücksetzen der Zeilennummer ist wichtig, damit der Dienst die Überwachung wieder an der richtigen Position aufnehmen kann. Die folgende Funktion übernimmt die Überprüfung der Logdatei:

Listing 4: Funktion zur Überprüfung von Logrotate

```
1 def checkLogRotate(self):
2     if os.path.isfile(self.file_path):
3         currentSize = os.stat(self.file_path).st_size
4         if currentSize < self.size:
5             self.last_position = 0
6         self.size = currentSize
7         return True
8     else:
9         return False
```

- **Schritt 3 Logdatei lesen und filtern**
  In diesem Schritt wird die Logdatei ab der letzten Position gelesen. Falls ein regulärer Ausdruck definiert wurde, werden die gelesenen Zeilen auf Übereinstimmung geprüft. Alle Zeilen, die nicht dem regulären Ausdruck entsprechen, werden von dem Dienst ignoriert. Wenn es neue Zeilen gibt, die von dem Dienst gefunden wurden, werden diese zusammen mit weiteren Informationen in ein JSON-Objekt transformiert und an das zentrale Überwachungssystem gesendet.

3.3.4 Überwachung von Systemparametern

Die Logdateiüberwachung ist geeignet, um die Funktionalität von Prozessen oder den Zugriff auf bestimmte Ressourcen zu überwachen. Die Logdateiüberwachung eignet sich jedoch nicht, um Systemparameter wie z.B. die CPU-Auslastung zu überwachen. Um solche Systemparameter zu überwachen wird eine andere Lösung benötigt. Der Dienst kann diese Aufgabe realisieren, indem er die Standardausgabe von ausgeführten Befehlen überwacht.

Der Thread der Klasse `MonitoringCommandThread` führt in dem vom Benutzer angegebenen Intervall einen Befehl aus und sendet die Standardausgabe an das zentrale Überwachungssystem. Wenn der Befehl eine fehlerhafte Syntax aufweist oder einen Rückgabewert ungleich 0 bei der Ausführung zurückgibt, beendet sich die Überwachung mit einer Fehlermeldung.

Für die Überwachung des freien Arbeitsspeichers eines Systems wäre zum Beispiel die Konfigurationszeile `c#free | sed '1d' | sed '2d' | awk '{print $2, $3}'#RAM#5000` geeignet. Der Befehl free gibt Informationen zum Arbeitsspeicher eines Systems zurück. Durch Filterung der Ausgabe und Spezifikation des Schlüsselworts `RAM` wird auf dem Dashboard ein Liniendiagramm angzeigt, das den Verlauf des Arbeitsspeichers des Systems zeigt.

3.3.5 Verfügbarkeitsüberwachung

Zusätzlich zu den in Abschnitt 3.1 angegebenen Anforderungen wurde ein Verfügbarkeitsüberwachung realisiert. Der Überwachungsserver überprüft und verwaltet zwei verschiedene von Überwachungsdaten. Das erste gibt die Verfügbarkeit eines Hosts und das zweite gibt die Verfügbarkeit der Überwachung an. Die Verfügbarkeitsüberwachung eines Hosts wird mit dem Kommandozeilenprogramm `ping` realisiert. Um zu überprüfen, ob die Überwachung eines Hosts aktiv ist, wird die Aktivität des verbundene Sockets getestet. Dies wurde realisiert, indem ein Timeout gesetzt wurde. Nach Ablauf des Timeouts wird überprüft, ob der Socket noch aktiv ist. Falls der Socket noch aktiv ist, wird der Timeout erhöht. Falls der Socket inaktiv ist, bedeutet das, dass keine Überwachung mehr aktiv ist. Ein festes Timeout kann nicht verwendet werden, da das Überwachungsintervall frei konfiguriert werden kann. Es ist lediglich denkbar, dass ein maximales Timeout spezifiziert

werden kann. In dieser Implementierung ist kein maximales Timeout vorgegeben, da dieses erst in einer realen Testumgebung abgeschätzt werden kann.

3.3.6 Nachrichtenformat des Dienstes

Der Dienst verpackt die Informationen in eine Zeichenkette im JSON-Format. Das dabei entstehende JSON-Objekt hat eine Ebene auf der sich alle Felder befinden. Der JSON-String kann folgende Felder enthalten:

- **Host:** Hostname des Systems
- **Time:** Timestamp in Sekunden von NTP(Network Time Protocol)-Servern aus Deutschland (de.pool.ntp.org)
- **Topic:** Eines vom Anwender konfiguriertes Signalwort zur Gruppierung und Steuerung der Ausgabe auf dem Dashboard.
- **Criticality:** Kritikalität der gefundenen Zeilen der Logdatei
- **Logfile_Name:** Pfad der Logdatei die überwacht wird.
- **Command:** Befehl der Ausgeführt wurde.
- **Result:** Ergebnis der Logdateiüberwachung oder eines Befehls.

Wenn die Nachricht von einer Logdateiüberwachung stammt, tauchen die Felder `Criticality` und `Logfile_Name` auf. Das Feld `Command` fehlt. Wenn die Nachricht von der Überwachung einer Befehlsausgabe stammt, existiert das Feld `Command`. Die Felder `Criticality` und `Logfile_Name` existieren nicht.

3.3.7 Implementierung der Überwachungsservers

Der Überwachungsserver ist von dem Dashboard unabhängig. Die einzige Aufgabe des Überwachungsservers ist es, die Überwachungsdaten zu empfangen und in die Redis Datenbank abzuspeichern. Das Dashboard entnimmt daraufhin Daten aus Redis und zeigt diese über die Webanwendung an. Die einzige Abhängigkeit die dabei entsteht, ist die Steuerung der Ausgabe auf dem Dashboard durch Schlüsselwörter. Die Abbildung 10 zeigt die Beziehung

zwischen dem Dashboard und dem Überwachungsserver auf. Der Überwachungsserver füllt Redis zum Beispiel mit dem Befehl `ZADD` mit Daten. Das Dashboard entnimmt mit dem Befehl `ZRANGE` die erforderlichen Daten für die Webanwendung.

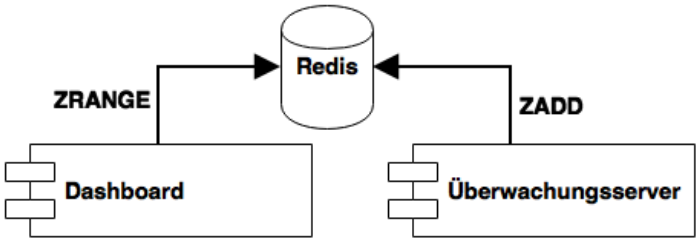


Abbildung 10: Datenfluss der Überwachungsservers

3.4 Sicherheit

In diesem Abschnitt werden die von dem Dienst umgesetzten Sicherheitsmaßnahmen genauer erleutert. Folgende Sicherheitsmaßnahmen, die bei der Sicherheitsbedarfsanalyse zusammengetragen wurden, wirken auf Anforderungen des Überwachungsdienstes:

Tabelle 3: Sicherheitsmaßnahmen die den Überwachungsdienst betreffen

Maßnahme	**Beschreibung**	**Wirkt auf Anforderung**
M01	Der Datenverkehr wird verschlüsselt	FA06
M03	Der Datenverkehr wird signiert	FA06

Der Dienst sendet systeminterne Daten über das Netzwerk. Diese Daten enthalten in der Regel vertrauliche Informationen über das System. Aus diesem Grund müssen diese Daten bei der Übertragung von dem System zum zentralen Überwachungsserver geschützt werden.

Die Lösung für dieses Problem ist die verschlüsselte Übertragung der Daten. Der Dienst setzt das hybride Verschlüsselungsprotokoll SSL ein, um die vertraulichen Daten an den zentralen Überwachungsserver zu senden. Durch Verwendung einer SSL-Verschlüsselung wird die Vertraulichkeit und

Integrität der Daten hergestellt. Im folgenden wird die Erzeugung eines SSL-Zertifikats mit OpenSSL und die Konfiguration des SSL-Servers genauer erleutert.

3.4.1 Erstellung eines SSL-Zertifikats

Bei der Erstellung der SSL-Zertifikate wurden die Richtlinien im OWASP Guide[10]verwendet. Demnach werden private Schlüssel mit 2048 Bit Länge verwendet. Dieser Schlüssel wird vor unauthorisierten Zugriff geschützt. Das Zertifikat wird nur von einem Server verwendet und hat einen qualifizierten Namen. Ein Domainname ist für das Zertifikat nicht notwendig. Die anderen Regeln werden nicht betrachtet, da sich diese auf Webanwendungen beziehen. Es wurde ein Server-Zertifikat und ein Client-Zertifikat erstellt. Das Client-Zertifikat wird von allen Clients zur Authentifizierung verwendet. Im folgenden wird das Vorgehen beschrieben, um das Server-Zertifikat zu erstellen.

1. Zunächst wird mit dem Befehl `genrsa` ein privater RSA Schlüssel erzeugt. Die Option `-des3` sorgt dafür, dass der erzeugte Schlüssel mit dem synchronen Verschlüsselungsverfahren Triple-DES(Data Encryption Standard) im CBC(Cipher Block Chaining Mode) Modus verschlüsselt wird. Die Zahl 2048 legt die Anzahl der Bits fest, aus die sich der private Schlüssel zusammensetzt.

```
1 openssl genrsa -des3 -out server.orig.key 2048
```

2. Dieser Befehl wandelt den Schlüssel in das PEM(Privacy Enhanced Mail) Format um.

```
1 openssl rsa -in server.orig.key -out server.key
```

3. Als nächstes wird mit dem Befehl 'req' aus dem Schlüssel eine CSR(Certificate Singning Request) erzeugt. Bei der Ausführung wird der Benutzer aufgeforderd Angaben zu dem Zertifikat einzugeben. Die Zertifikatsregistrierungsanforderung könnte einer Zertifizierungsstelle zur digitalen Unterschrift vorgelegt werden.

[10]`https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Server_Certificate`

```
openssl req -new -key server.key -out server.csr
```

4. Da dieses Zertifikat nur zur Kommunikation zwischen Systemen verwendet wird, ist eine digitale Unterschrift von einer Zertifizierungsstelle nicht erforderlich. Deswegen wird das Zertifikat mit dem Befehl `x509` selbst Unterschrieben. Die Option `-days` legt fest, wieviele Tage das Zertifikat gültig ist. Das Zertifikat server.crt wird von dem Dienst für die Verschlüsselung des Datenverkehrs verwendet.

```
openssl x509 -req -days 365 -in server.csr
   -signkey server.key -out server.crt
```

5. Das zentrale Überwachungssystem ist in Java implementiert. Java verwendet das `keytool`, um mit Zertifikaten zu arbeiten. Zertifikate müssen mit dem „keytool“ in einen „keystore“ importiert werden, bevor man diese im Programm verwenden kann. Das Client-Zertifikat kann ohne Umwege direkt importiert werden. Damit der „keystore“ das mit OpenSSL erzeugte Server-Zertifikat und den privaten Schlüssel importieren kann, muss dieses zuerst in das PKCS#12-format umgewandelt werden. Der Befehl `pkcs12` erzeugt aus dem X.509 Zertifikat und dem privaten Schlüssel die erforderliche Datei im PKCS#12-format.

```
openssl pkcs12 -export -in server.crt
   -inkey server.key -chain -CAfile server.crt
   -name "monitoringDaemon" -out server.p12
```

3.4.2 Konfiguration des SSL-Servers

Bevor der Überwachungsserver SSL-Verbindungen entgegennimmt, wird diese Konfiguriert. Bei der Konfiguration der SSL-Verbindung wurde der OWASP Guide[(11)]verwendet.

Die Methode `configureSSL()` der Klasse `MonitoringDaemonServer` nimmt ein Objekt des Typs `ServerSocket` entgegen und gibt ein konfiguriertes Objekt des Typs `SSLServerSocket` zurück. Der nachfolgende Code zeigt einen Ausschnitt dieser Methode.

[(11)]`https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Server_Protocol_and_Cipher_Configuration`

Listing 5: Konfiguration des SSL-Servers

```
1  private SSLServerSocket configureSSL(ServerSocket socket) {
2    SSLServerSocket configuredSocket = (SSLServerSocket)socket;
3    SSLParameters sslParameters = configuredSocket.getSSLParameters();
4    // only support TLSv.1.1 and TLSv1.2
5    ArrayList<String> protocolsToSet = new ArrayList<>();
6    for(String protocol: configuredSocket.getSupportedProtocols())
7    {
8      if(supportedProtocols.contains(protocol))
9      {
10       protocolsToSet.add(protocol);
11     }
12   }
13 ...
14   // only support special set of secure ciphers
15   ArrayList<String> ciphersToSet = new ArrayList<>();
16   for(String cipher: configuredSocket.getSupportedCipherSuites())
17   {
18     if(supportedCiphers.contains(cipher))
19     {
20       ciphersToSet.add(cipher);
21     }
22   }
23 ...
24   sslParameters.setUseCipherSuitesOrder(true);
25   sslParameters.setNeedClientAuth(true);
26   configuredSocket.setSSLParameters(sslParameters);
27   return configuredSocket;
28 }
```

Es werden nur die Protokolle „TLSv.1.1“ und „TLSv1.2“ verwendet. Die SSL-Versionen 1, 2 und 3 gelten nach OWASP als unsicher und sollten nicht mehr verwendet werden. Um die unterstützten Cipher Suites zu bestimmen, wurde die Liste der von Java unterstützten Cipher Suites[(12)]untersucht. Aufgrund von Importregulierungen[(13)]enthält diese Liste nur bestimmte Cipher Suites. Anschließend wurde diese Liste gefiltert und sortiert, sodass alle Regeln von OWASP eingehalten werden. Daher können nur die Cipher Suites von dem Server verwendet werden, die in nachfolgender List aufgeführt sind. Die Reihenfolge der Cipher Suites ist die Reihenfolge, in der diese von oben nach

[(12)]http://docs.oracle.com/javase/8/docs/technotes/guides/security/SunProviders.html

unten aufgeführt sind.

Listing 6: Sichere SSL-Cipher Suites

```
1  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
2  TLS_RSA_WITH_AES_128_GCM_SHA256
3  TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
4  TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
5  TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
6  TLS_RSA_WITH_AES_128_CBC_SHA256
7  TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
8  TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
9  TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
10 TLS_RSA_WITH_AES_128_CBC_SHA
11 TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
12 TLS_DHE_RSA_WITH_AES_128_CBC_SHA
13 TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
14 TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
15 TLS_EMPTY_RENEGOTIATION_INFO_SCSV
```

3.5 Installation und Konfiguration des Dienstes

Der Überwachungsdienst muss auf den zu überwachenden Systemen installiert und konfiguriert werden. In diesem Abschnitt wird erklärt wie dabei vorgegangen werden muss. Es wird an einem Beispielsystem gezeigt, wie eine Konfiguration des Dienstes erfolgen könnte.

3.5.1 Installation als Linux-Dienst

1. **Systemvorraussetzungen überprüfen:** Als erstes sollte sichergestellt werden, dass Python in der Version 2.x oder 3.x auf dem System installiert ist. Bei den meisten Linux-Betriebssystemen ist Python bereits vorinstalliert. Um zu überprüfen, ob Python bereits installiert ist, kann folgender Befehl verwendet werden:

   ```
   1 which python && which python3
   ```

   Die Version von Python kann entweder mit dem Befehl `python --version` oder `python3 --version` überprüft werden.

[13] `https://docs.oracle.com/javase/8/docs/technotes/guides/security/SunProviders.html#importlimits`

Wenn Python noch nicht installiert ist, kann dies über das Packetverwaltungsprogramm APT mit den Befehl
`sudo apt-get install python` oder
`sudo apt-get install python3` erledigt werden. Wenn kein Packetverwaltungsprogramm existiert, kann die Installation von Python auch manuell erfolgen. Dafür müssen die Quelldateien[(14)] heruntergeladen werden und anschließend kompiliert werden.

2. **Dienst ausführen und konfigurieren:** Um den Dienst zu starten, muss je nachdem welche Python Version installiert ist, der Befehl
`python MonitoringDaemon.py` oder
`python3 MonitoringDaemon.py` ausgeführt werden. Zuvor muss der Dienst über die Datei `monitor.conf` konfiguriert werden. Der Dienst sollte nicht mit Root-rechten gestartet werden. Es muss vorher bestimmt werden, welche Rechte der Dienst benötigt um seine Überwachungsaufgaben durchzuführen. Ziel sollte hier sein dem Dienst so wenig Berechtigungen zu geben, um gerade noch seine Aufgabe erfüllen zu können(least privilege principle). Dies kann zum Beispiel über eine ACL(Access Control List), die einem bestimmten Benutzer zugeordnet ist, realisiert werden.

3. **Dienst über Init-Prozess starten und stoppen:** Da das System in der Regel immer im angeschalteten Zustand überwacht werden soll, ist es zu empfehlen den Dienst in den Init-Prozess einzubinden. Dadurch wird der Dienst bereits beim Systemstart ausgeführt. Es gibt mehrere Implementierungen von Init-Systemen(z.B. System V-Init, Upstart oder Systemd). Im folgenden wird für das „System V-Init“ System ein Beispielskript angegeben.

    - **System V-Init:** Das System V-Init ist ein init-System. Der erste Prozess der von dem Kernel gestartet wird ist der Init-Prozess mit der Prozess-ID 1. Dieser Prozess startet daraufhin die einzelnen Dienste. Wie die einzelnen Dienste gestartet werden sollen, wird in sogenannten Init-Skripte festgelegt. Diese Skripte befinden sich alle unter „/etc/init.d/“. Das folgende Listing zeigt ein Beispiel, wie das Init-Skript für den Überwachungsdienst auszusehen hat.

[(14)] `https://www.python.org/downloads/source/`

Listing 7: SysVinit-Skript zum starten und stoppen des Überwachungsdienstes

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          MonitoringDaemon
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: starts and stops the MonitoringDaemon
# Description:       starts and stops the MonitoringDaemon
### END INIT INFO

SCRIPTPATH=/pathToTheScript
SCRIPTNAME=MonitoringDaemon.py
PIDFILE="/tmp/monitoringDaemon.pid"
SCRIPTPARGUMENTS=""

case "$1" in
  start)
    echo "Starting␣MonitoringDaemon"
    # Start the service
    echo $SCRIPTPATH/$SCRIPTNAME $SCRIPTARGUMENTS
    python $SCRIPTPATH/$SCRIPTNAME $SCRIPTARGUMENTS > \
      /dev/null 2> /dev/null &
    ;;
  stop)
    echo "Stopping␣MonitoringDaemon"
    # Stop the service using the pid-file written during __init__
    kill -2 `cat $PIDFILE`
    ;;
  restart)
    echo "Restarting␣MonitoringDaemon"
    kill -2 `cat $PIDFILE`
    sleep 5
    python $SCRIPTPATH/$SCRIPTNAME $SCRIPTARGUMENTS > \
      /dev/null 2> /dev/null &
    ;;
  *)
    echo "Usage:␣/etc/init.d/MonitoringDaemon␣{start|stop|restart}"
    exit 1
    ;;
esac

exit 0
```

3.5.2 Installation als Docker-Container

1. **Systemvorraussetzungen überprüfen:** Es gibt nur eine einzige Vorrausetzung die das System erfüllen muss, um den Dienst als Docker-Container zu starten. Auf dem System muss die Docker-Engine installiert sein. Wie Docker auf dem System installiert werden muss kann der Dokumentation[15]entnommen werden.

2. **Dienst konfigurieren:** Der Dienst muss vor der Erstellung des Docker-Images aus dem Dockerfile konfiguriert werden. In dem Verzeichnis, in dem sich das Dockerfile befindet, gibt es ein weiteres Unterverzeichnis mit dem Namen `appFiles`. Dort befindet sich die Datei `monitor.conf` mit der die Konfiguration des Dienstes durchgeführt werden kann.

3. **Docker-Image aus Dockerfile erzeugen:** Als nächstes wird aus dem Dockerfile ein Image erzeugt. Mit dem Befehl

```
1 docker build -t cs.hm.edu.shm.monitoring.service .
```

   kann aus dem Dockerfile ein Image erzeugt werden, welches das Tag `cs.hm.edu.shm.monitoring.service` besitzt.

4. **Docker-Container starten:** Jetzt muss der Docker-Container mit dem folgenden Befehl gestartet werden.

```
1 docker run --net=host --restart=always
2   [--volume=/host-dir:/container-dir:ro]
3   d --name cs.hm.edu.shm.redis.monitoring.service
4   -t cs.hm.edu.shm.monitoring.service:latest
```

   Die Option `--restart=always` sorgt dafür, dass der Dienst nach einem Absturz wieder gestartet wird.
   Die Optionen `--volume=/host-dir:/container-dir:ro` müssen bei einer Logdateiüberwachung angegeben werden, damit die Logdatei von dem Container zugegriffen werden kann. Diese Option und die Konfiguration des Dienstes müssen genau abgestimmt werden, damit der Dienst korrekt auf die Logdateien zugreifen kann.

[15] `https://docs.docker.com/engine/installation/`

5. **Container über Init-Prozess starten und stoppen:** Wie bereits bei der Installation als Init-Dienst beschrieben, muss ein Init-Skript erstellt werden. Dieses Initstkript muss lediglich den Docker-Container starten.

3.5.3 Beispielkonfiguration des Dienstes

Um ein besseres Verständnis für die Konfiguration des Dienstes zu erlangen, werden in diesem Abschnitt einige Beispiele dagelegt. Die vorgestellten Konfigurationszeilen sind von dem verwendeten System Ubuntu 14.04 und den Programmversionen abhängig. Diese dienen daher lediglich als Beispiele und müssen für jedes System angepasst werden.

1. **Ram-Last überwachen:** Der Befehl `free` kann verwendet werden, um den freien und belegten Arbeitsspeicher des Systems anzuzeigen. Somit ist dieser Befehl ideal geeignet, um die RAM-Last eines Systems zu überwachen. Damit das Dashboard die Ram-Last grafisch formatiert anzeigt, muss bei der Konfiguration das 3. Feld auf „ram“ gesetzt werden. Darüberhinaus muss die Ausgabe des Befehls noch formatiert werden. Folgende Konfigurationszeile ist zum Beispiel geeignet, um mit der Programmversion `procps-ng 3.3.9` von `free` die Ram-Last zu überwachen:

```
1 c#free | sed '1d' | sed '2,$d' | \
2   awk '{print $2, $3}'#RAM#5000
```

2. **CPU-Last überwachen:** Die CPU-Auslastung kann mit dem Befehl `mpstat -P ALL` ausgegeben werden. Wie bereits bei der Überwachunng der Ram-Last erwähnt wurde, muss die Ausgabe noch formatiert werden und das 3. Feld auf „CPU“ gesetzt werden, um eine formatierte Ausgabe auf dem Dashboard zu erreichen. Folgende Konfigurationszeile kann mit der Version `sysstat version 10.2.0` von „mpstat“ zur Überwachung der CPU-Last verwendet werden:

```
1 mpstat -P ALL | sed '1,3d' | awk '{print $2, $12}'
```

3. **Festplattenspeicher überwachen:** Der freie Speicherplatz auf der Festplatte, kann mit dem Befehl `df -h` angezeigt werden. Die Ausgabe muss wie bei den Beispielen zuvor noch formatiert werden und das 3.

Feld auf „hdd“ gesetzt werden, damit das Dashboard eine formatierte Ausgabe erzeugt. Die Konfigurationszeile lautet:

```
c#df -h | grep '^/dev/[hs]d' | \
  awk '{s+=$2; p+=$3} END {print s, p}'#HDD#5000
```

Es wurde die Programmversion „8.21“ von df verwendet.

4. **Fehlgeschlagene SSH-Logins überwachen:** In der Datei `/var/log/auth.log` werden alle Versuche protokoliert sich an das System anzumelden. Dazu zählen auch Anmeldungen über SSH. Das bedeutet, dass eine Überwachung dieser Datei sinnvoll wäre, um fehlgeschlagene Anmeldeversuche zu überwachen. Bei einer Logdateiüberwachung werden alle eintreffenden Meldungen überwacht. Das bedeutet, dass es auch zu einer Meldung kommt, wenn lediglich eine erfolgreiche Anmeldung eines Benutzers durchgeführt wurde. Um dies zu verhindern kann ein reguläre Ausdruck bei der Konfiguration angegeben werden. Mit den beiden folgenden Konfigurationszeilen kann die Überwachung von fehlgeschlagenen Anmeldeversuchen durchgeführt werden.

```
l#/var/log/auth.log#logfile#10000#WARNING#.*sshd.*Failed.*
l#/var/log/auth.log#logfile#10000#WARNING#.*authentication failure.*
```

3.6 Evaluation

In diesem Kapitel wird der entwickelte Überwachungsdienst evaluiert. Es wurde eine Evaluation der SSL-Verschlüsselung durchgeführt.

3.6.1 Evaluation der SSL-Verschlüsselung

In diesem Abschnitt wird die Evaluation der SSL-Verschlüsselung erleutert. Es wurde ein Black-Box-Test durchgeführt, dabei ist das Vorgehen dem OWASP Guide[16]angelehnt.

1. **Port-scanning mit nmap:** Ein Angreifer würde vermutlich zunächst die offenen Ports des Systems scannen, um einen Angriffspunkt zu finden. Es gibt bereits viele Werkzeuge, um Port-scanning durchzuführen.

[16]https://www.owasp.org/index.php/Testing_for_SSL-TLS_(OWASP-CM-001)#Black_Box_Test_and_example

Zu den bekanntesten Werkzeuge zählen „Nessus[17]“ und „nmap[18]“. Für die Evaluation wurde das Kommandozeilenwerkzeug nmap verwendet. Die nachfolgende Ausgabe zeigt den durchgeführten Port-scan.

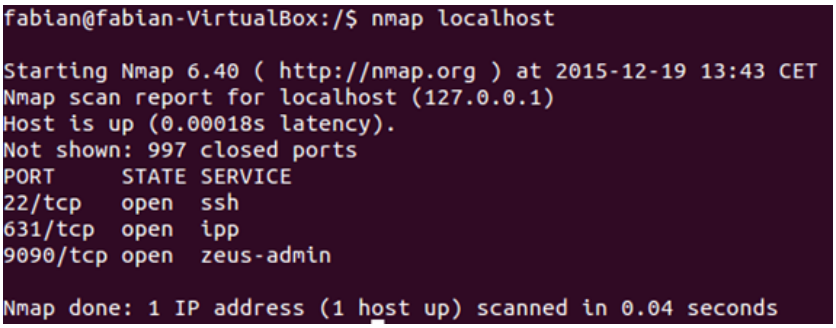


Abbildung 11: Port-scanning mit nmap

Es gibt drei offene Ports. Der Port 22 wird für eingehende SSH-Verbindungen verwendet und der Port 631 ist für Drucker reserviert. Der Port 9090 ist der Überwachungsserver, der eine SSL-Verbindung auf Port 9090 entgegennimmt.

2. **Überprüfe SSL-Protokoll mit OpenSSL:** OpenSSL[19]bietet das Kommandozeilenwerkzeug „s_client“ an, um einen einfachen Client zu simulieren. Dieses Werkzeug kann zum Beispiel dazu verwendet werden, um einen SSL-Server zu testen, welche Protokolle er unterstützt. Dadurch kann erkannt werden, ob der SSL-Server eine SSL-Verbindungen mit einem unsicheren Protokoll eingeht. Das nachfolgende Bild zeigt einen Test, ob der SSL-Server eine SSL-Verbindung mit den von OWASP für unsicher angegebenen Protokollen SSL und TLSv1.0 annimmt.

[17] `http://www.tenable.com/products/nessus-vulnerability-scanner`
[18] `https://nmap.org/`
[19] `https://www.openssl.org/`

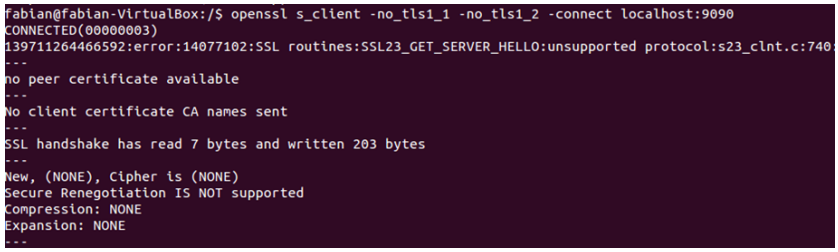


Abbildung 12: Testen des SSL-Servers mit s_client

Die Ausgabe zeigt einen Fehler. Das Bedeutet, dass von dem Server keine SSL-Protokolle und das Protokoll TLSv1.0 nicht unterstützt wird.

3. **sslScan:** Das Kommandozeilenwerkzeug sslScan[20] überprüft die unterstützten SSL-Ciphersuites des Servers und markiert die unsicheren Ciphersuites. Die nachfolgende Ausgabe zegt einen Ausschnitt des durchgeführten Back-Box Tests mit sslScan.

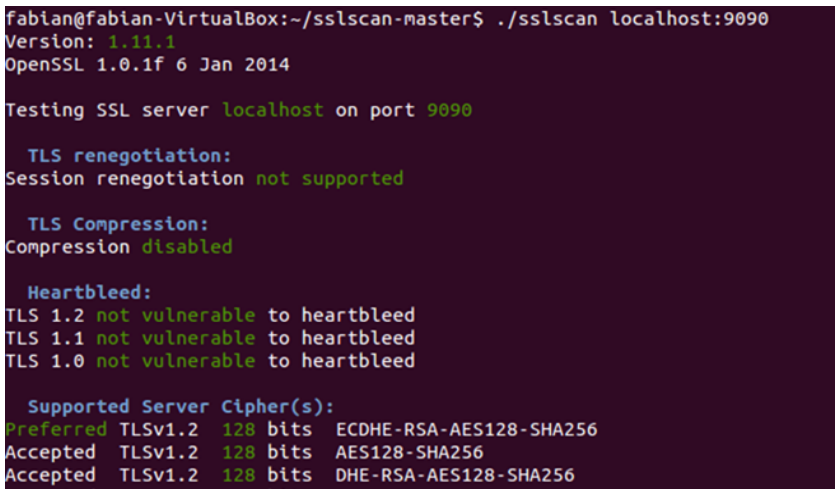


Abbildung 13: Testen des SSL-Server mit sslScan

Die Evaluation hat ergeben, dass keine unsicheren Ciphersuites von dem SSL-Server unterstützt werden.

[20] https://github.com/rbsec/sslscan

4. **Wireshark:** Um den Ablauf des SSL-Handshakes zu evaluieren, wurde das Werkzeugt Wireshark [21]verwendet. Der nachfolgende Sceenshot zeigt den Ablauf einer SSL-Verbindung mit dem Überwachungsserver.

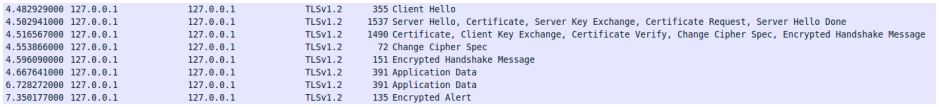


Abbildung 14: Evaluation des Protokollverlaufs mit wireshark

In dem Mitschnitt ist ersichtlich, dass ein korrekter SSL-Handshake durchgeführt wird anschließend zwei Datenpakete versendet werden und die Verbindung wieder geschlossen wird. Die Analyse des Datenverkehrs hat außerdem gezeigt, dass der Server ein Client-Zertifikat anfordert, um eine Verbindung aufzubauen.

3.6.2 Integrationstest mit dem Dashboard

Um zu überprüfen, ob der Überwachungsdienst die Anforderungen erfüllt, wurde ein Integrationstest zusammen mit dem Dashboard durchgeführt. Dabei wurde der Überwachungsdienst mit verschiedenen Konfigurationsszenarien ausgeführt. Anschließend wurde die resultierende Ausgabe auf dem Dashboard betrachtet.

3.7 Fazit und Ausblick

Es wurde ein Überwachungsdienst implementiert, der viele verschiedene Aspekte eines Systems überwachen kann. Dabei wurde der Schwerpunkt auf die Vertraulichkeit und Integrität der Datenübertragung gelegt. Zusätzlich dazu wurde auf eine dynamische Konfiguration gesetzt, damit der Überwachungsdienst für möglichst viele Anwendungsfälle verwendet werden kann. Über den in Abschnitt 3.1 angegebenen Anforderungen hinaus wurde eine Verfügbarkeitsüberwachung realisiert. Der Überwachungsdienst kann noch durch einige Aspekte erweitert werden. Diese werden im folgenden Aufgezählt.

[21]`https://www.wireshark.org/`

- Im Moment wird der Vorgang eines Logrotates von dem Überwachungsdienst entdeckt. Daraufhin wird die zuletzt gelesene Zeile der Logdatei korregiert. Es existieren jedoch noch Szenarien, bei denen durch ein Logrotate Überwachungslücken auftreten. Diese Überwachungslücken könnten in einer zukünftigen Version geschlossen werden.

- Bei Meldungen der Systemüberwachung, die mit „CRITICAL“ eingestuft sind, könnte der Überwachungsserver eine mit S/MIME (Secure / Multipurpose Internet Mail Extensions) verschüsselte E-Mail an einen vorgegebenen Verteiler versenden.

- Es wird zwar eine Verfügbarkeitsüberwachung von dem Überwachungsserver durchgeführt, jedoch werden noch keine Meldungen versendet, wenn ein kritischer Systemzustand eintritt In einer zukünftigen Version könnte der Überwachungsserver eine Meldung versendet, falls das System nicht mehr Verfügbar ist oder die Systemüberwachung ausfällt.

- Im dieser Version verwenden alle Clients das selbe Zertifikat Eine sicherere Variante wäre es, wenn jeder Client ein eigenes Zertifikat benötigt, um mit dem Überwachungsserver kommunizieren zu können Dadurch wäre, falls durch einen Angriff der private Schlüssel bekannt wird nicht die komplette Kommunikation gefährdet.

- Aufgrund von Importregulierungen verwendet der Server nicht die stärksten Verschlüsselungsalorithmen die Verfügbar sind. Die SSL-Konfiguration des Überwachungsservers könnte nach Installation der kryptografischen Erweiterung der Verschlüsselungsalgorithmen[(22)]erneut durchgeführt werden.

[(22)]`http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html`

4 Dashboard

4.1 Einleitung

Neben dem Monitoring Daemon zur Akquirierung von Daten und dem ELK Stack (Elasticsearch, Logstash und Kibana) ist das Dashboard eine weitere zentrale Komponente des Systems SichHeimMonitor.
Abbildung 15 zeigt einen ersten groben Entwurf der Systemarchitektur, der in den folgenden Abschnitten um Details ergänzt und verfeinert wird.

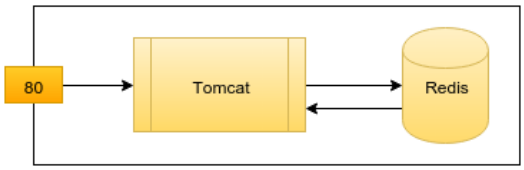


Abbildung 15: Erster Entwurf der Systemarchitektur

orange	Port
gelb	Prozess/Datei

4.2 Funktionale Anforderungen

Die in Java entwickelte Webapplikation soll die folgenden funktionalen Anforderungen besitzen:

Funktionale Anforderung	Beschreibung
FA01	Der Zugriff auf die Applikation soll über ein gesichertes, offenes und standardisiertes Protokoll erfolgen.
FA02	Die Applikation soll über einen Bereich verfügen der nur für authentifizierte und autorisierte Benutzer zugänglich ist.
FA03	Die Applikation soll es Benutzern ermöglichen, eine Browser Session zu starten, in der keine Daten erhoben werden und beim Beenden vernichtet werden.
FA04	Die Applikation soll in der Lage sein, sowohl lokale Systemparameter, als auch entfernte Dienste zu Überwachen.
FA05	Die Applikation soll in der Lage sein, laufende Docker Container anzuzeigen und diese zu steuern (Starten, Stoppen, Neustarten)
FA06	Die Applikation soll in der Lage sein, die Anzahl der versuchten Logins zu speichern.

4.3 Sicherheitskritische Punkte der Architektur

Aus den funktionalen Anforderungen und dem ersten Entwurf der Architektur ergeben sich die folgenden Sicherheitskritischen Punkte der Anwendung:

Problem	Beschreibung
P01	Große Angriffsfläche durch unterschiedliche Komponenten
P02	Keine Authentifizierung innerhalb der Applikation
P03	Keine Autorisierung innerhalb der Applikation
P04	Verwendung unsicherer Kommunikation
P05	Keine Authentifizierung an der Datenbank (Redis)
P06	Datenbank unterstützt keine verschlüsselte Speicherung

4.4 Maßnahmen und Lösungen

Für mögliche Lösungen werden die in der Schutzbedarfserstellung definierten Maßnahmen zugrunde gelegt.

Maßnahme	Beschreibung
M01	Verschlüsselte Datenübertragung
M02	Verschlüsselte Speicherung der Daten
M03	Signierte Datenübertragung
M04	Benutzerauthentifizierung über OAuth
M05	Replizierung wichtiger Dienste
M06	Härtung und Konfiguration der einzelnen Komponenten

Betrachtet man nun die funktionalen Anforderungen zusammen mit den Kritischen Punkten und den Maßnahmen der Schutzbedarfsfestellung ergeben sich die folgenden Lösungen:

Lösung	Beschreibung
L01	Isolation und Härtung der Anwendung durch Verwendung von Docker
L02	Authentifizierung gegenüber github.com mittels OAuth
L03	Autorisierung durch Nutzung der github.com Funktionalität von Organisationen und Teams
L04	Die Kommunikation der Anwendung erfolgt über HTTPS
L05	Authentifizierung an der Datenbank
L06	Verschlüsselte Speicherung der akquirierten Daten

4.5 Architektur

Der folgende Abschnitt beschreibt die verfeinerte Architektur des Systems unter Berücksichtigung der im vorherigen Abschnitt ermittelten Probleme und Lösungen

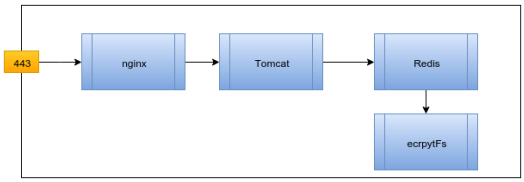


Abbildung 16: Verfeinerter Entwurf der Systemarchitektur

blau	Docker Container
orange	Port
gelb	Prozess/Datei

4.5.1 L01 Isolation und Härtung der Anwendung durch Verwendung von Docker

Der folgende Abschnitt beschreibt die sichere Konfiguration der einzelnen Komponenten aus dem vorherigen Abschnitt. Dabei wird nicht auf globale Sicherheitskonzepte des Betriebssystems (z.b. Firewall, verschlüsseltes Dateisystem, etc. . .) eingegangen, sondern nur auf Konzepte auf Applikationsebene.

1. **Sichere Konzepte für alle Komponenten**
   Der folgende Abschnitt gibt einen Überblick über Konzepte die für alle Komponenten (Tomcat und Redis) verwendet wurden.

   (a) **Verwenden einer stabilen Version**
   Es wurde sichergestellt, dass eine stabile Version der jeweiligen Komponente verwendet wurde. Eine stabile Version ist in der Regel die Version die über keine bekannten Sicherheitslücken verfügt und ausreichend getestet wurde.
   Um über eventuelle Schwachstellen informiert zu werden, sollte man sich auf der entsprechenden Mailingliste registrieren und im Ernstfall Sicherheitsupdates installieren.
   Von der Verwendung der aktuellsten Version wird in der Regel abgeraten, da diese weniger getestet und anfälliger für Sicherheitslücken sind.

   (b) **Dedizierten Benutzer und Gruppe verwenden**
   Für den Betrieb der Komponenten sollte ein eigener Benutzer und eine dedizierte Linuxgruppe mit minimalen Berechtigungen verwendet werden.
   Der Besitzer des Installationsverzeichnis sollte der dedizierte Benutzer und die Gruppe sein.

   (c) **Deployment als Docker Container**
   Obwohl es nicht zwingend erforderlich ist, werden die Komponenten aus Abschnitt [4.5] innerhalb eines Docker Containers bereitgestellt. Dies hat gegenüber einer herkömmlichen Installation, folgende Vorteile:

   - Rapid application deployment:

Container beinhalten in der Regel nur die minimal benötigten Abhängigkeiten zur Laufzeit, was zu einer reduzierten Größe und einem schnelleren deployment führt.

- Portabilität:

Eine Anwendung und all ihre Abhängigkeiten können in einem einzelnen Container zusammengeführt werden, der unabhängig ist vom Host des Containers. Der Container kann auf andere Systeme, auf denen ebenfalls Docker läuft, transferiert und ausgeführt werden, ohne das Kompatibilitätsprobleme entstehen.

- Isolation:

Docker Container laufen unabhängig voneinander in einer eigenen Sandbox. Der Zugriff auf andere Container erfolgt über definierte Schnittstellen.

- Versionskontrolle:

Container können versioniert sein, so dass es einfach möglich ist, Änderungen nachzuvollziehen und bei Bedarf rückgängig zu machen.

- Teilen von Container:

Docker bietet die Möglichkeit, Container in Remote Repositories mit anderen zu teilen und so wiederzuverwenden.

2. **Absicherung von nginx**
   Für nginx wird auf den in Abschnitt [5.3.1] beschriebenen Docker Container des ELK Stack zurückgegriffen.

3. **Absicherung von Redis**
   Der folgende Abschnitt bietet einen Überblick über die sichere Konfiguration von Redis. Die einzelnen Punkte beziehen sich größtenteils auf die offizielle Installationsanleitung [15]

   (a) **Eingeschränkter Zugriff**
   Redis ist so konzipiert, das der Zugriff von vertrauenswürdigen Clients innerhalb einer vertrauenswürdigen Umgebung erfolgen sollte. Es ist deshalb in der Regel nicht empfehlenswert, Redis direkt über das öffentliche Netz verfügbar zu machen.

Um dies zu erreichen wird Redis als Docker Container bereitgestellt und der Port nicht nach außen freigegeben. Dadurch ist es möglich, dass nur andere Docker Container mit der Datenbank kommunizieren können.
Eine weitere Möglichkeit besteht darin, den Zugriff auf ein bestimmtest Netzwerkinterface zu beschränken. Dazu ist der folgende Eintrag in der Datei redis.conf notwendig.

```
1 bind 127.0.0.1
```

Aufgrund der dynamischen IP Zuweisung von Docker und der Tatsache dass die Redis Konfiguration keine Subnetze verwalten kann, wird der erste Weg umgesetzt.

(b) **Authentifizierung**
Redis besitzt kein Access Control Feature (z.B. über Rollen und Rechte), allerdings eine optionale Konfiguration zur Authentifizierung. Sobald die Konfiguration aktiviert ist, verweigert Redis die Funktionalität für nicht authentifizierte Clients.
Ein Client kann sich authentifizieren, in dem er das vom Administrator gesetzte Passwort übermittelt. Da Redis auf Geschwindigkeit bei der Verarbeitung von Anfragen optimiert ist, sollte das Passwort lange genug sein, um nicht mittels Brute Force Angriff geknackt werden kann.
Der folgende Eintrag ist hierfür in der redis.conf notwendig:

```
1 requirepass e2f7ef60-9c24-11e5-8994-feff819cdc9f
```

(c) **Ausschalten nicht benötigter Kommandos**
Redis bietet eine Vielzahl von Kommandos, die entweder komplett deaktiviert werden oder umbenannt werden können. Ein gefährlicher Befehl ist z.B. FLUSHALL welcher die komplette Datenbank löscht. Um einen Befehl umzubenennen ist folgender Eintrag in der Datei redis.conf notwendig:

```
1 rename-command FLUSHALL \
2    b840fc02d524045429941cc15f59e41cb7be6c52
```

Danach ist der Befehl FLUSHALL nur noch über
b840fc02d524045429941cc15f59e41cb7be6c52 ausführbar.
Will man den Befehl komplett deaktivieren muss folgendes in die Konfiguration eingetragen werden

```
rename-command FLUSHALL ""
```

Neben dem Befehl FLUSHALL wurden keine weiteren Kommandos deaktiviert.

(d) **Dockerfile**
Listing 8 zeigt das Dockerfile zur Erzeugung eines auf Alpine Linux basierendem Docker Image dass die im vorherigen Abschnitt beschriebene Konfiguration umsetzt.

Listing 8: Dockerfile

```
FROM alpine:3.2
MAINTAINER Tobias Placht <info@tobiasplacht.de>
RUN \
apk add --update build-base linux-headers

RUN addgroup redis && \
adduser -S redis && \
adduser redis redis
RUN \
cd /tmp && \
wget http://download.redis.io/redis-stable.tar.gz && \
tar xvzf redis-stable.tar.gz && \
cd redis-stable && \
make && \
make install && \
mkdir -p /etc/redis && \
rm -rf /tmp/redis-stable*
COPY config/redis.conf /etc/redis/redis.conf
RUN mkdir /data && touch /data
RUN chown -R redis:redis /data
VOLUME ["/data"]
WORKDIR /data
USER redis
CMD ["redis-server", "/etc/redis/redis.conf"]
```

Dabei wird ausgehend von Alpine Linux, zunächst die zur Kompilierung von Redis benötigten Compiler und Linux-Kernel-Header installiert.
Anschließend wird ein unprivilegierter Benutzer samt zugehöriger Gruppe erstellt und die stabile Version von Redis aus dem Quellcode installiert.
Als letzter Schritt werden definierte Konfigurationsdateien an die

benötige Stelle kopiert und die Berechtigungen gesetzt.
Mittels dem Befehl

```
1 docker build -t cs.hm.edu.shm.redis .
```

wird das Image erzeugt und mit dem Tag cs.hm.edu.shm.redis zur weiteren Verwendung versehen.

4. **Absicherung von Tomcat**
   Der folgende Abschnitt bietet einen Überblick über die sichere Konfiguration von Tomcat. Dabei wird nach dem Artikel Securing Tomcat der OWASP vorgegangen. [16]

   (a) **Entfernen nicht benötigter Features**
   Tomcat kommt standardmäßig mit zusätzlichen, in der Regel nicht benötigter Features, um die Administration zu vereinfachen. Dabei handelt es sich um die in der nachfolgend angegebenen Tabelle:

Name	**Beschreibung**	**Benötigt**
DOCS	Serverdokumentation	nein
EXAMPLES	Ausführbare Beispielprojekte	nein
HOST-MANAGER	Graphische Verwaltung virtueller Hosts	nein
MANAGER	Graphischer Verwaltung der Java Applikationen der Server Instanz	nein
ROOT	Beispielapplikation	nein

   Durch jede weitere Applikation erhöht sich die potenzielle Angriffsfläche, die verwendeten Ressourcen und der Wartungsaufwand. Ebenfalls kritisch, ist der Umstand, dass die Funktionen in der Standardeinstellung alle den selben Benutzernamen und Passwort verwenden.
   Um die nicht benötigten Funktionen zu entfernen, müssen folgende Verzeichnisse und Dateien entfernt werden. CATALINA_HOME bezeichnet dabei den Ordner, in dem Tomcat installiert wurde.

```
1 rm -rf CATALINA_HOME/webapps/*
2 rm -rf CATALINA_HOME/server/webapps/*
3 rm -rf CATALINA_HOME/conf/Catalina/localhost/host-manager.xml
```

```
4 rm -rf CATALINA_HOME/conf/Catalina/localhost/manager.xml
```

(b) **Entfernen der Versionsnummer aus Fehlerseiten**
In der Standardeinstellung, enthalten Tomcat Fehlerseiten, die Versionsnummer des Server. Dieses Verhalten erhöht die Angriffsfläche, da so gezielt nach Exploits und Schwachstellen gesucht werden kann, die für diese Version bekannt sind.
Um das Verhalten zu ändern, muss die Datei org/apache/catalina/util/ServerInfo.properties wie folgt angepasst werden.:

```
1 vorher: server.info=Apache Tomcat 8.30
```

```
1 nachher: server.info=Apache Tomcat
```

(c) **Ausschalten des Stacktrace**
Die Standard Fehlerseite zeigt den Stacktrace der Java Applikation an. Um dieses Verhalten zu unterbinden muss die Datei web.xml wie folgt angepasst werden:

```
1 <error-page>
2   <exception-type>java.lang.Throwable</exception-type>
3   <location>/error.jsp</location>
4 </error-page>
```

(d) **Ersetzen des Server String**
Es ist möglich, den HTTP Header anzupassen, den der Server als Antwort an eine Anfrage sendet. Dazu muss die Datei server.xml wie folgt angepasst werden. Anschließend identifiziert sich der Server dem Client nicht mehr als Tomcat sondern Apache.

```
1 <Connector port="8080" server="Apache" />
```

(e) **Absichern des Shutdown Ports**
Es besteht die Möglichkeit, den Server über den Port 8005 herunterzufahren, indem das Shutdown Command gesendet wird. In der Standardeinstellung ist der Befehl dazu Shutdown. Es wird empfohlen das Kommando durch eine zufällige, schwer zu erratende Zeichenkette zu ersetzen bzw. ganz zu entfernen.
Zum ändern muss die server.xml wie folgt geändert werden:

```
1 <Server port="8005" shutdown="ReallyComplexWord">
```

Zum entfernen muss die Datei CATALINA_HOME/conf/server.xml wie folgt geändert werden:

```
<Server port="-1" shutdown="ReallyComplexWord">
```

(f) **Dockerfile**
Listing 9 zeigt das Dockerfile zur Erzeugung eines auf Alpine Linux basierendem Docker Image dass die im vorherigen Abschnitt beschriebene Konfiguration sicher stellt.

Listing 9: Dockerfile

```
FROM java:8-jdk
ENV CATALINA_HOME /usr/local/tomcat
ENV PATH $CATALINA_HOME/bin:$PATH
RUN mkdir -p "$CATALINA_HOME"
WORKDIR $CATALINA_HOME
RUN \
gpg --keyserver pool.sks-keyservers.net --recv-keys\
05AB33110949707C93A279E3D3EFE6B686867BA6 \
07E48665A34DCAFAE522E5E6266191C37C037D42 \
47309207D818FFD8DCD3F83F1931D684307A10A5 \
541FBE7D8F78B25E055DDEE13C370389288584E7 \
61B832AC2F1C5A90F0F9B00A1C506407564C17A3 \
79F7026C690BAA50B92CD8B66A3AD3F4F22C4FED \
9BA44C2621385CB966EBA586F72C284D731FABEE \
A27677289986DB50844682F8ACB77FC2E86E29AC \
A9C5DF4D22E99998D9875A5110C01C5A2F6059E7 \
DCFD35E0BF8CA7344752DE8B6FB21E8933C60243 \
F3A04C595DB5B6A5F1ECA43E3B7BBB100D811BBE \
F7DA48BB64BCB84ECBA7EE6935CD23C10D498E23

RUN groupadd tomcat
RUN useradd -g tomcat -d $CATALINA_HOME tomcat

ENV TOMCAT_MAJOR 8
ENV TOMCAT_VERSION 8.0.30
ENV TOMCAT_TGZ_URL https://www.apache.org/dist/tomcat/\
    tomcat-$TOMCAT_MAJOR/v$TOMCAT_VERSION/bin/\
    apache-tomcat-$TOMCAT_VERSION.tar.gz

RUN set -x \
&& curl -fSL "$TOMCAT_TGZ_URL" -o tomcat.tar.gz \
&& curl -fSL "$TOMCAT_TGZ_URL.asc" -o tomcat.tar.gz.asc \
&& gpg --verify tomcat.tar.gz.asc \
&& tar -xvf tomcat.tar.gz --strip-components=1 \
```

```
35 && rm bin/*.bat \
36 && rm tomcat.tar.gz*
37
38 # OWASP: Remove everything from WEBAPPS folder
39 RUN ["rm", "-rf", "/usr/local/tomcat/webapps"]
40
41 # OWASP: Remove Version String
42 RUN cd $CATALINA_HOME/lib && \
43 /usr/bin/jar xf catalina.jar \
44 org/apache/catalina/util/ServerInfo.properties && \
45 sed -i 's@Apache␣Tomcat/8.0.30@Apache␣Tomcat@' \
46 org/apache/catalina/util/ServerInfo.properties && \
47 /usr/bin/jar uf catalina.jar \
48 org/apache/catalina/util/ServerInfo.properties && \
49 rm -rf org/apache/catalina/util/ServerInfo.\
50 propertiesorg/apache/catalina/util/ServerInfo.properties
51
52 COPY dashboard-webapp-spark.war \
53 /usr/local/tomcat/webapps/ROOT.war
54 COPY keystore.jks /usr/local/tomcat/keystore.jks
55
56 # OWASP Config
57 RUN ["rm", "-rf", "/usr/local/tomcat/conf/server.xml"]
58 COPY server.xml /usr/local/tomcat/conf/server.xml
59
60 # OWASP Config
61 RUN ["rm", "-rf", "/usr/local/tomcat/conf/web.xml"]
62 COPY web.xml /usr/local/tomcat/conf/web.xml
63
64 RUN chown -R tomcat:tomcat $CATALINA_HOME
65 USER tomcat
```

4.5.2 L02/L03 Authentifizierung und Autorisierung

Da keine globale Authentifizierung vorhanden ist, die die einzelnen Projekte nutzen können, wird für das Dashboard eine Authentifizierung gegenüber github.com implementiert. Der Zugriff auf die API und die auf github.com hinterlegten Daten erfolgt mittels OAuth.
Prinzipiell kann die Authentifizierung auch gegen andere Instanzen erfolgen, allerdings ist kein in Deutschland angesiedelter Anbieter bekannt. Wünschenswert wären vertrauenswürdige Instanzen beispielsweise Hochschulen oder andere Deutsche Behörde, die sich an geltendes Recht, insbesondere Datenschutz, halten.

1. **OAuth**
   OAuth ist ein offenes, standardisiertes (RFC 6749) Protokoll zur sicheren Autorisierung innerhalb von Desktop, Mobile und Web-Applikationen. [17] Die aktuelle Version ist 2.0, welche inkompatibel zum Vorgänger 1.0 ist.
   OAuth bietet Client-Applikationen, einen sicheren und delegierten Zugriff auf Server Ressourcen im Auftrag des Ressourceneigentümers, ohne dem Client die Zugangsdaten zur Verfügung zu stellen.
   OAuth wurde speziell für die Verwendung mit HTTP entwickelt. Zu den bekannten OAuth Anbietern gehören Facebook, Google, GitHub und Twitter.
   Im Vergleich zur Basic Authentication mittels Benutzername und Passwort, verwendet OAuth einen Token basierten Mechanismus, mit zwei entscheidenden Vorteilen:

   - Revocable Access: Der Zugriff kann jederzeit vom User widerrufen werden
   - Limited Access: Benutzer können die Zugriffsrechte überprüfen, die ein Token erhält, bevor sie eine Applikation autorisieren.

2. **GitHub**
   GitHub ist ein Internet basierender git repository hosting Service. Neben den von Git bekannten Funktionalitäten, bietet es weitere exklusive Features an.
   Stand 2015, besitzt GitHub über 11 Millionen Benutzer und über 29 Millionen Git repostiories. Dadurch handelt es sich bei GitHub um den weltweit größten Hoster der Welt.

3. **GitHub API**
   Die aktuelle Version zum Zeitpunkt dieses Dokumentes ist Version v3. Jeglicher Zugriff auf die API erfolgt über HTTPS über die URL api.github.com. Sämtliche Daten werden dabei sowohl als JSON als gesendet und empfangen.

```
1 $ curl https://api.github.com/zen
2 Practicality beats purity.
```

   GitHub stellt sowohl öffentliche Details (Avatar-url, Name, Location, etc. . .) als auch private Informationen der einzelnen Benutzer über die

API zur Verfügung.

```
$ curl https://api.github.com/users/knacht
```

```
{
  "login": "knacht",
  "id": 1736327,
  "avatar_url": \
  "https://avatars.githubusercontent.com/u/1736327?v=3",
  "gravatar_id": "",
  ...
  "type": "User",
  "site_admin": false,
  "name": "Tobias␣",
  "company": "HSWT",
  "blog": null,
  "location": "Munich,␣Germany",
  "email": null,
  "hireable": true,
  "bio": null,
  "public_repos": 7,
  "public_gists": 1,
  "followers": 2,
  "following": 1,
  "created_at": "2012-05-14T06:55:09Z",
  "updated_at": "2015-11-07T12:14:07Z"
}
```

Sofern sich ein Benutzer mittels Basic Authentication oder OAuth autorisiert, ist es möglich seine eigenen privaten Details zu empfangen.

```
{ "private_gists": ,
  "total_private_repos": ,
  "owned_private_repos": ,
  "disk_usage": ,
  "collaborators": ,
  "plan": {
    "name": ,
    "space": ,
    "collaborators": ,
    "private_repos":
  }
}
```

4. **Web Application Flow für Drittanbieter**
   Damit ein Drittanbieter (in diesem Falle SichHeimMonitor) Zugriff auf die Daten eines GitHub Benutzers erhält muss er zunächst eine Applikation registrieren.

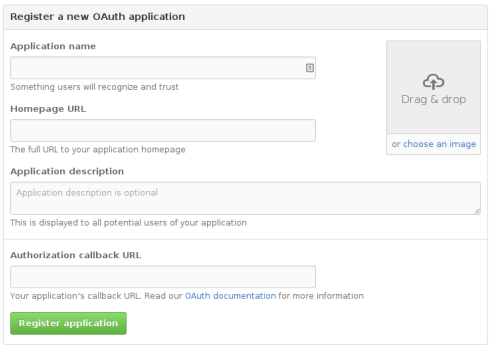


Abbildung 17: Registrierung einer GitHub Application

Anschließend wird der folgende Flow durchlaufen:

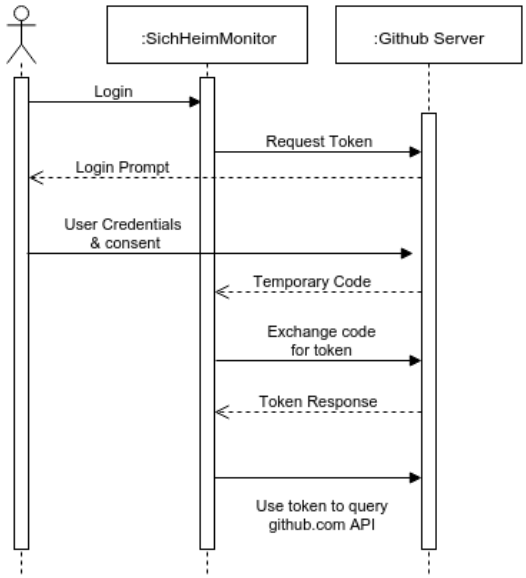


Abbildung 18: GitHub Wep Application Flow

Der Benutzer will sich innerhalb der Applikation einloggen. Die Applikation leitet den Benutzer weiter an GitHub. Dort gibt der Benutzer seinen Benutzernamen und Password ein und akzeptiert die Berechtigungen. GitHub antwortet mit einem temporären Code, den die Applikation gegen einen Token austauscht. Mit diesem Token ist es dann möglich, die GitHub API zu benutzen bis:

(a) Sich der Benutzer bei GitHub ausloggt

(b) Der Benutzer den Token revidiert

(c) Die Applikation weitere Rechte fordert

5. **Autorisierung mithilfe von GitHub Organisation**
   GitHub.com ermöglicht es Benutzern, teil einer oder mehrerer Organisationen zu sein. Die Organisationen eines Benutzers lassen sich über die API abfragen, sofern dieser dem zugestimmt hat.
   Für das Projekt SichHeimMonitor wurde eine Organisation erstellt, in der sich die Teammitglieder befinden.

6. **Zugriffskontrolle**
   Zusammen mit der Authentifizierung und Autorisierung gegenüber github.com ist innerhalb der Applikation, folgendes Rechtesystem realisiert:

   (a) Der Benutzer ist anonym, d.h. er ist nicht über github.com angemeldet. Dadurch besteht nur die Möglichkeit, einen anonymen Tor Browser über Docker zu starten, der nach dem Beenden des Fensters gelöscht wird.

   (b) Der Benutzer ist mit seinem github.com Account angemeldet und nicht Teil der Organisation SichHeimMonitor. Er besitzt dadurch eingeschränkten Zugriff und kann innerhalb der Applikation, nur sein github.com Profil ansehen.

   (c) Der Benutzer ist mit seinem github.com Account angemeldet und Teil der Organisation SichHeimMonitor. Er besitzt dadurch vollen Zugriff und kann alle Funktionalitäten der Applikation verwenden. Um Mitglied der Organisation zu werden, ist ein persönliches Gespräch mit den bestehenden Mitglieder notwendig.
   Die beschriebene Zugriffskontrolle kann beliebig angepasst werden und dient nur der Demonstration.

4.5.3 L04 Sichere Kommunikation durch HTTPS

Um Vertraulichkeit und Integrität während der Kommunikation zu gewährleisten, erfolgt jegliche Kommunikation, sowohl innerhalb, als auch außerhalb des Systems über HTTPS. Abbildung 19 zeigt die Stellen des Systems an denen einen sichere Kommunikation über HTTPS statt finden soll.

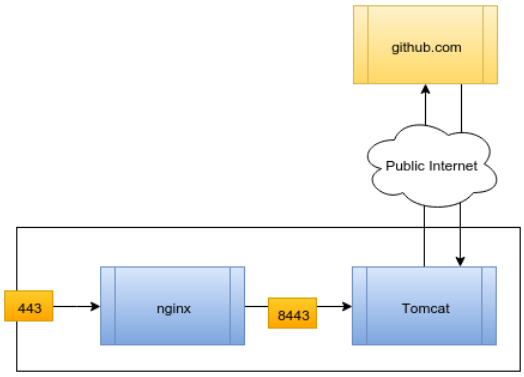


Abbildung 19: Architektur mit sicherer Kommunikation

1. **Selbst signiertes HTTPS Zertifikat**
   Durch die Verwendung des in Abschnitt [5.3.1] vorgestellten nginx ist sichergestellt, dass der Tomcat Server nicht direkt erreichbar ist, sondern nur über eine HTTPS Verbindung.
   Während der Entwicklung und für den Fall dass der Reverse Proxy ausfällt, wurde mit OpenSSL ein selbst signiertes Zertifikat erstellt um die Kommunikation über HTTPS zu gewährleisten.
   Dazu wird im ersten Schritt ein privater Schlüssel erzeugt. Der Parameter aes256 legt dabei das zur Erzeugung verwendete Verschlüsselungsverfahren fest. In diesem falle AES (Advanced Encryption Standard) mit einer Schlüssellänge von 256 Bit. Der Parameter 4096 legt die Länge des privaten Schlüssels fest.

```
1 openssl genrsa -aes256 -out server-key.pem 4096
```

   Anschließend wird ein neues, selbst signiertes Zertifikat erstellt. Der Parameter -days 365 gibt an wie lange das Zertifikat gültig ist. In diesem falle also für ein Jahr.

```
openssl req -new -x509 -days 365 -key server-key.pem \
    -sha256 -out server.pem
```

Während der Erstellung können die folgenden Angaben gemacht werden:

```
Country Name (2 letter code)
State or Province Name (full name)
Locality Name (eg, city)
Organization Name (eg, company)
Organizational Unit Name
Common Name (e.g. server FQDN or YOUR name)
Email Address []
```

Für die weitere Verwendung innerhalb von Tomcat muss das Zertifikat in das PKCS12 Format umgewandelt werden. Dies ist ebenfalls mit OpenSSL möglich.

```
openssl pkcs12 -export -out keystore.p12 \
    -inkey server-key.pem -in server.pem
```

Zur Nutzung innerhalb des Tomcat Application Server muss ein Java Keystore erstellt werden. Hierfür stellt Java das Programm keytool zur Verfügung. Der folgende Befehl erzeugt einen neuen Keystore im JKS Format.

```
keytool -importkeystore -destkeystore keystore.jks \
    -srcstoretype PKCS12 -srckeystore keystore.p12
```

Als letzter Schritt muss noch ein Tomcat Connector konfiguriert werden. Dieser benötigt den Port, den Keystore und das dazugehörige Passwort. Listing zeigt den relevanten Ausschnitt der Konfiguration. Die verfügbaren Ciphers und das verwendete Protokoll folgen dabei der Empfehlung der OWASP aus dem Artikel Securing Tomcat. [16] Dabei wurde eine Konfiguration mit geringerer Kompatibilität, aber erhöhter Sicherheit gewählt.

```
<Connector port="8443"
  protocol="org.apache.coyote.http11.Http11NioProtocol"
  maxThreads="150" SSLEnabled="true" scheme="https"
  secure="true"
  clientAuth="false" sslProtocol="TLSv1.2"
  ...
```

```
  server="Apache"
  sslEnabledProtocols="TLSv1.2"
  ciphers="TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
␣␣TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
␣␣TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384,
␣␣TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384,
␣␣TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
␣␣TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
␣␣TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256,
␣␣TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256,
␣␣TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,
␣␣TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,
␣␣TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
␣␣TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384,
␣␣TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384,
␣␣TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,
␣␣TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
␣␣TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
␣␣TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
␣␣TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256,
␣␣TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256,
␣␣TLS_ECDH_RSA_WITH_AES_128_CBC_SHA,
␣␣TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA" />
```

2. **Kommunikation mit github.com**
   Die github.com API ist nur über HTTPS zu erreichen. Der für Tomcat konfigurierte Keystore überschreibt den Standard Java Keystore der sonst implizit verwendet wird. Deshalb ist es notwendig, den default keystore in seinen eigenen keystore zu importieren. Dies erfolgt über das Java Keytool und dem folgenden Befehl.

```
keytool -importkeystore \
    -srckeystore /usr/java/jdk1.8.0_65/jre/lib/security/cacerts \
    -destkeystore keystore.jks -srcstorepass changeit \
    -deststorepass MySecretPassword
```

   Anschließend erfolgt die Kommunikation über github.com bei Verwendung eines eigenen Keystores ebenfalls mit HTTPS
   Für den Fall dass github.com nicht verfügbar ist, besitzt die Applikation einen Fallback Modus, der der klassischen Basic Authentication mit Benutzername und Passwort entspricht. Benutzername und Passwort sind nur den Administratoren bekannt.

4.5.4 L05 Signierte und verschlüsselte Kommunikation mit dem Docker Daemon

Docker verwendet eine Client-Server Architektur wie in Abbildung 20 zu sehen.

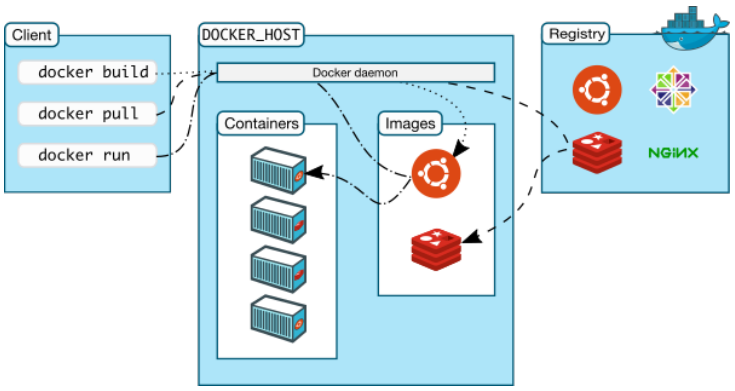


Abbildung 20: Docker Client Server Modell [23]

In dieser kommuniziert der Docker Client mit dem Docker Daemon welcher die eigentliche Arbeit übernimmt. Der Client und Server können entweder auf dem selben System oder zwei unterschiedlichen Systemen installiert sein. Die Kommunikation zwischen Client und Server erfolgt entweder über Sockets oder RESTful API. Um beispielsweise zu überprüfen, ob der Server erreichbar ist kann folgende GET Anfrage gesendet

```
1 GET /_sing
```

Falls der Server erreichbar ist, erhält der Client folgende Antwort:

```
1 HTTP/1.1 200 OK
2 Content-Type: text/plain
3
4 OK
```

1. **Docker API konfigurieren**
   In der Standardeinstellung ist der Docker Daemon nur per UNIX Socket

[23] https://docs.docker.com/engine/introduction/understanding-docker/

erreichbar. Dies hat zur Folge, dass die API inkl. dem Client nur vom Benutzer root, bzw. Mitglieder der Gruppe `docker` nutzbar ist. Um die API für Java zugänglich zu machen, muss zunächst der Port freigegeben werden. Mit folgendem Befehl, ist der Docker Daemon für jedermann per HTTP über den Port 4243 erreichbar.

```
/usr/bin/docker daemon -H=0.0.0.0:4243
```

Diese Einstellung stellt eine extreme Sicherheitslücke dar und sollte unter keinen Umständen konfiguriert werden. Dadurch ist es beispielsweise möglich, sich mittels der folgenden Abfrage eine Liste aller Container zu erhalten und diese anschließend zu löschen.

```
GET /containers/json?all=1
```

2. **Docker API sicher konfigurieren**
   Es besteht die Möglichkeit, den Docker Server so zu konfigurieren, dass die Kommunikation ausschließlich über HTTPS und zertifizierte Clients zugänglich ist.

   Dazu muss im ersten Schritt, eine Certificate Authority(CA) erstellt werden. Dies kann ähnlich wie bei der Konfiguration von Tomcat über OpenSSL erfolgen. Für die CA wird ein privater 4096 Bit AES 256 Schlüssel erzeugt.

```
openssl genrsa -aes256 -out ca-key.pem 4096
```

   Anschließend wird ein Zertifikat erstellt das mit dem privaten Schlüssel signiert ist.

```
openssl req -new -x509 -days 365 -key ca-key.pem \
    -sha256 -out ca.pem
```

   Im nächsten Schritt wird ein Server Schlüssel und certifcate signing request erstellt, welche mit der CA signiert wird. Dazu sind die folgenden SSL Befehle notwendig:

```
# Server key erstellen
openssl genrsa -out server-key.pem 4096
# CSR erstellen
openssl req -subj "/CN=$HOST" -sha256 -new \
    -key server-key.pem -out server.csr
# Erlaubte IP Adressen festlegen
```

```
echo subjectAltName = IP:10.10.10.20,IP:127.0.0.1 > \
    extfile.cnf
# Server Key signieren
openssl x509 -req -days 365 -sha256 -in server.csr \
    -CA ca.pem -CAkey ca-key.pem \
    -CAcreateserial -out server-cert.pem \
    -extfile extfile.cnf
```

Für zertifizierte Clients ist ein ähnliches vorgehen notwendig. Da die Webapplikation, der einzige Client ist, der mit dem Docker Daemon kommuniziert ist ein einzelnes Zertifikat ausreichend. Das Client Zertifikat wurde wie folgt erstellt:

```
# Client key erstellen
openssl genrsa -out key.pem 4096
# CSR erstellen
openssl req -subj '/CN=client' -new \
    -key key.pem -out client.csr
# Client Authentication konfigurieren
echo extendedKeyUsage = clientAuth > \
    extfile.cnf
# Client Key signieren
openssl x509 -req -days 365 -sha256 \
    -in client.csr -CA ca.pem \
    -CAkey ca-key.pem \
    -CAcreateserial -out cert.pem \
    -extfile extfile.cnf
```

Mit einer Standard `umask` von 022 sind die privaten Schlüssel für jeden lesbar und veränderbar. Um diese zu schützen werden die Zugriffsrechte wie folgt angepasst

```
chmod -v 0400 ca-key.pem key.pem server-key.pem
```

Anschließend kann der Docker daemon mit den folgenden Parametern neu gestartet werden

```
docker daemon --tlsverify --tlscacert=ca.pem \
    --tlscert=server-cert.pem --tlskey=server-key.pem \
    -H=0.0.0.0:2376
```

Danach erlaubt Docker nur noch Verbindungen von Clients die ein von der CA signiertes Zertifikat besitzen. Sobald die Zertifikat basierte Authentifizierung aktiviert ist, benötigt Docker keine root rechte mehr. Es

ist deshalb darauf zu achten, dass Zertifikat wie ein Root Passwort zu betrachten und dementsprechend damit umzugehen.
Die Verbindung kann unter Linux beispielsweise mit Curl erfolgen:

```
curl https://$HOST:2376/_ping \
  --cert ~/.docker/cert.pem \
  --key ~/.docker/key.pem \
  --cacert ~/.docker/ca.pem
```

3. **Docker Client Zertifikat unter Java benutzen**
   Um das im vorherigen Abschnitt erstellte Client Zertifikat unter Java zu benutzen, muss ein Trustore angelegt werden. Dabei kann auf das `keytool` zurückgegriffen werden, welches schon zur Erstellung des Keystore für den Tomcat Server verwendet wurde.
   Im ersten Schritt muss zunächst das als pem vorliegende Client Zertifikat in das PKCS12 Format umgewandelt werden.

```
openssl pkcs12 -export -out truststore.p12 \
    -inkey key.pem -in cert.pem
```

   Anschließend müssen das Client Zertifikat, das Server Zertifikat und das CA in einen gemeinsamen trustore importiert werden.

```
# Client Zertifikat importieren
keytool -importkeystore -destkeystore truststore.jks \
    -srcstoretype PKCS12 -srckeystore truststore.p12
# Server Zertifikat importieren
keytool -importcert -trustcacerts -file server-cert.pem \
    -keystore truststore.jks
# CA importieren
keytool -importcert -trustcacerts -file ca.pem \
    -keystore truststore.jks -alias ca
```

   Um den so erzeugten Truststore zu benutzen, ist folgende Tomcat Konfiguration notwendig

```
<Connector port="8443" \
    protocol="org.apache.coyote.http11.Http11NioProtocol"
    maxThreads="150" SSLEnabled="true" scheme="https"
    secure="true"
    clientAuth="false" sslProtocol="TLSv1.2"
```

```
6       keystoreFile="/usr/local/tomcat/keystore.jks"
7       keystorePass="MySecretPassword"
8       keystoreFile="/usr/local/tomcat/truststore.jks"
9       keystorePass="MyTrustStorePassword"
10      server="Apache"
11      sslEnabledProtocols="TLSv1.2"
12      ciphers="TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
13  ␣␣␣␣␣␣TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
14  ␣␣␣␣␣␣TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384,
15  ␣␣␣␣␣␣TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384,
16  ␣␣␣␣␣␣TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
17  ␣␣␣␣␣␣TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
18  ␣␣␣␣␣␣TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256,
19  ␣␣␣␣␣␣TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256,
20  ␣␣␣␣␣␣TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,
21  ␣␣␣␣␣␣TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,
22  ␣␣␣␣␣␣TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
23  ␣␣␣␣␣␣TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
24  ␣␣␣␣␣␣TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384,
25  ␣␣␣␣␣␣TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384,
26  ␣␣␣␣␣␣TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,
27  ␣␣␣␣␣␣TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,
28  ␣␣␣␣␣␣TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
29  ␣␣␣␣␣␣TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,
30  ␣␣␣␣␣␣TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
31  ␣␣␣␣␣␣TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
32  ␣␣␣␣␣␣TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256,
33  ␣␣␣␣␣␣TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256,
34  ␣␣␣␣␣␣TLS_ECDH_RSA_WITH_AES_128_CBC_SHA,
35  ␣␣␣␣␣␣TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA" />
```

4.5.5 L06 Verschlüsselte Speicherung der Login Daten

Für die Speicherung der akquirierten Daten wird auf das in Abschnitt [5] beschriebene Verfahren zurückgegriffen.

4.5.6 Implementierung

Der folgende Abschnitt gibt einen Überblick über ausgewählte Implementierungsdetails der Webapplikation.

1. **Speicherung der Login versuche**
   Abbildung [21] zeigt die Indexseite die für jeden erreichbar ist.

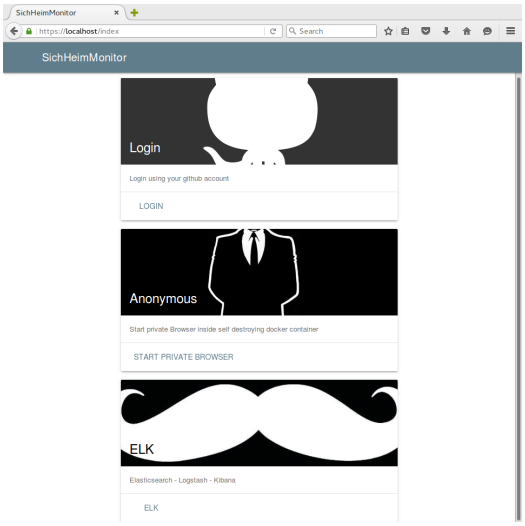


Abbildung 21: Dashboard Index

Die Anzahl der Versuchten Logins wird mithilfe eines Sorted Sets in Redis gespeichert. Bei einem sortiertem Redis Set handelt es sich um eine Sammlung von eindeutigen Strings die mit einem Score versehen werden können. Existiert die Kombination aus Key und Member noch nicht, wird diese mit einem Score von 1 angelegt, ansonsten der Score um den Wert 1 erhöht.
Mit folgendem Befehl wird dem der Score, des Member „one‘ünter dem Key „myzset‘üm 1 erhöht.

```
1 ZINCRBY myzset 1 "one"
```

Nach einem Klick auf die Login Komponente wird ein Redis Sorted Set mit folgendem Key und Member erstellt und in die Datenbank eingefügt.

```
1 # Key
2 visits-by-url:/login:09-01-2016
3 # Member
4 16:05
```

Dazu dient der folgende Programmcode.

```
@Override
public void insertIntoCurrentHourAndMinute(String path) {
    try (Jedis jedis =
            JedisConnectionPool.INSTANCE.getResource()) {
        jedis.zincrby(getKey(path), 1, getHourAndMinute());
    }
}

private String getKey(String path) {
    return PREFIX + path + ":" + getDate();
}

private String getDate() {
    LocalDateTime now = LocalDateTime.now();
    int year = now.getYear();
    int month = now.getMonthValue();
    int day = now.getDayOfMonth();
    return String.format("%02d-%02d-%02d",
        day, month, year);
}

private String getHourAndMinute() {
    LocalDateTime now = LocalDateTime.now();
    int hour = now.getHour();
    int minute = now.getMinute();
    return String.format("%02d:%02d",
        hour, minute);
}
```

Der vordere Teil des Key ist statisch und lautet "visits-by-url". Über den Parameter path der Methode getKey kann prinzipiell die zu überwachende URL angegeben werden. Der letzte Teil des Keys ist das aktuelle Datum im Format TT-MM-YYYY. Als Member dient die aktuelle Stunde und Minute im Format HH:MM
Anschließend ist das Set in Redis wie folgt hinterlegt:

```
# Query for last five Member for Key
# visits-by-url:/login:09-01-2016 with scores
zrange visits-by-url:/login:09-01-2016 -1 -1 WITHSCORES
 1) "13:29"
 2) "2"
```

D.h. am 09.01.2016 erfolgten die meisten Zugriffe um 13:29, nämlich zwei Stück.

2. **Abfrage und Visualisierung der versuchten Logins**
   Um die Zugriffe abzufragen, kann wie im vorherigen Abschnitt die Funktion zrange verwendet werden. Mit den Parametern `start` und `end` kann festgelegt werden, wieviele Member abgefragt werden sollen. Um alle abzufragen ist folgender Befehlt notwendig

```
zrange visits-by-url:/login:09-01-2016 0 -1
```

Um die Ergebnisse einzugrenzen, kann das Intervall von $START bis -1 erfolgen, wie in folgendem Programmcode zu sehen. Dabei hat der Parameter Start den Wert -10, d.h. es werden die 10 häufigsten Zugriffe abgefragt. Die Ergebnisse die als String vorliegen, werden abgebildet auf Objekte der Klasse TimeSeries. Das Ergebnis wird anschließend anhand des Timestamp sortiert und in einer Liste gesammelt.

```
@Override
public List<TimeSeries> getTopTenMember(String path) {
    try (Jedis jedis =
            JedisConnectionPool.INSTANCE.getResource()) {
        String key = getKey(path);
        return jedis
    .zrange(key, -10, -1)
    .stream()
    .map(s -> new TimeSeries(key + s,
            jedis.zscore(key, s)))
        .sorted((e1, e2) -> e1.getTimeStamp()
        .compareTo(e2.getTimeStamp()))
        .collect(Collectors.toList());
    }
}
// Getter and setter ommited
public class TimeSeries implements Serializable {

private static final long serialVersionUID = 1L;

private String timeStamp;

private Double value;

```

```
25 public TimeSeries(String timeStamp, Double value) {
26       this.timeStamp = timeStamp;
27       this.value = value;
28 }
```

Abbildung 22 zeigt die Anzahl der Login Versuche die mit einem Skript erzeugt wurden.

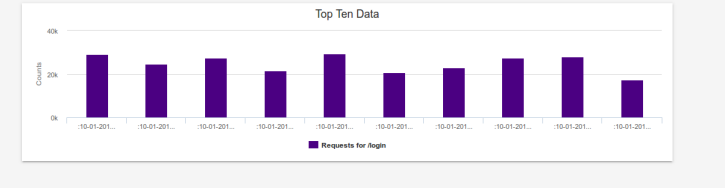


Abbildung 22: Visualisierung versuchter Logins

Die Datenstruktur ist so entwickelt, dass sie sowohl für weitere URLs als auch Zeiträume anwendbar ist. Mit dem Redis Befehl ZUNIONSTORE ist es möglich einzelne Tage zu Monaten zu vereinen. Konkret implementiert ist aktuell allerdings nur die Top 10 Daten für die URL Login.

3. **Überwachen lokaler Systemparameter**
   Das Dashboard wird dazu verwendet um lokale Systemparameter des Hosts anzuzeigen.

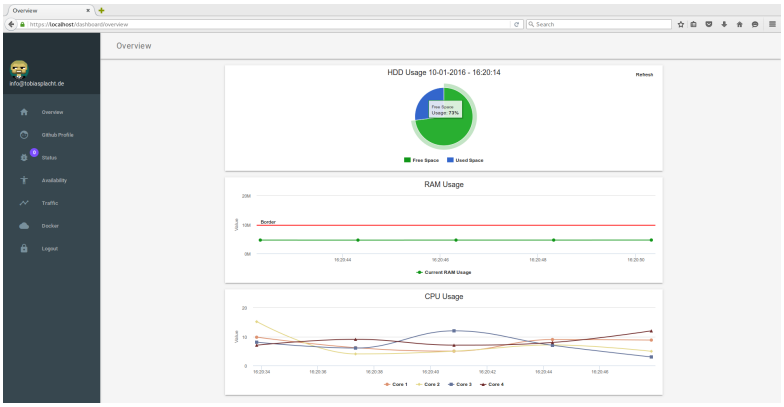


Abbildung 23: Überwachung lokaler Systemparameter

Dazu wir auf dem Host der Monitoring Daemon mit folgender Konfiguration als Docker Container installiert:

```
# Monitoring server settings
IP:mds
Port: 9090

# Monitoring configuration
c#df -h | grep '^/dev/[hs]d' | \
    awk '{s+=$2; p+=$3} END {print s, p}'#HDD#10000
c#free | sed '1d' | sed '2d' | \
    awk '{print $2, $3}'#RAM#2000
c#mpstat -P ALL 1 1| sed '1,14d' | \
    awk '{print $2, $12}'#CPU#2000
```

Die Daten wurde vom Monitoring Daemon Server in Redis gespeichert und vom Dashboard auf der Übersichtsseite abgefragt, ausgewertet und für die Visualisierung aufbereitet.
Für die Implementierung wurden die folgenden drei Parameter mit den dazugehörigen Metriken definiert

Parameter	Status INFO	Status WARNING	Status CRITICAL
CPU Auslastung in Prozent	< 80	< 90	> 90
RAM Verbrauch in Prozent	< 60	< 70	> 70
Freier Speicher aller HDDs	< 80	< 90	> 90

Der Folgende Codeausschnitt zeigt die Umsetzung der Metriken als Java Enum

```
public enum LocalRating {

    RAM("RAM", 60, 70),
    CPU("CPU", 80, 90),
    HDD("HDD", 80, 90);

    LocalRating(String key,
            int thresholdInfo,
            int thresholdWarning) {
        this.key = key;
        this.thresholdInfo = thresholdInfo;
```

```
12          this.thresholdWarning = thresholdWarning;
13  }
```

Mittels

```
1   @Override
2   public Set<String> getStatus(LocalRating status) {
3       String key = "";
4       try {
5           key = InetAddress.getLocalHost().getHostName()
6               + ":" + status.getKey();
7       } catch (UnknownHostException e) {
8     LOGGER.error("An␣error␣occured");
9       }
10      try (Jedis jedis =
11              JedisConnectionPool.INSTANCE.getResource()) {
12          return jedis.zrange(key, -1, -1);
13      }
14  }
```

wird der Status aus der Datenbank abgefragt, und beispielsweise für den RAM wie folgt ausgewertet:

```
1   @Override
2   public Object handle(Request request, Response response)
3           throws Exception {
4       String commandString = getCommand(LocalRating.RAM);
5
6       if (commandString.isEmpty()) {
7           return null;
8       }
9       Command command =
10          gson.fromJson(commandString, Command.class);
11      String result = command.getResult();
12      String[] splitted = result.split("␣");
13
14      long total = Long.parseLong(splitted[0]);
15      long used = Long.parseLong(splitted[1].trim());
16
17      long percentage = (used * 100) / total;
18
19      writeStatusIntoDb(LocalRating.RAM,command, percentage);
20
21      return new RamStatus(total, used);
```

```
22
23 }
```

Der ausgewertete Wert wird in die Datenbank mit dem dazugehörigen Status gewertet und angezeigt

4. **Kommunikation mit der Docker API**
   Das Dashboard ist in der Lage folgende Docker Informationen abzufragen:

   - Auslesen der Docker Version
   - Auslesen der Docker API Version
   - Auslesen der Git Commit ID
   - Auslesen der Go Version
   - Auslesen des Host OS inkl. Architektur und Kernel Informationen
   - Auflisten aller Images und dazugehörigen Details
   - Auflisten aller Container und dazugehörigen Details inkl. laufender Prozessen
   - Starten, Stoppen und Neustarten einzelner Container

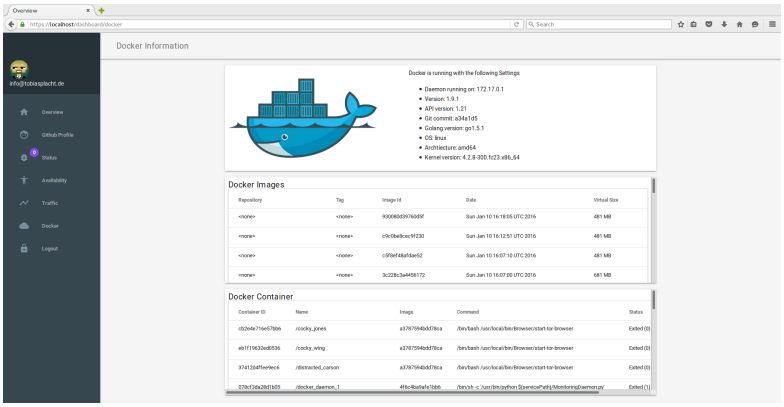


Abbildung 24: Übersicht Docker

Die Informationen werden dabei über die abgesicherte API abgefragt und sind durch folgendes Interface definiert:

```
public interface DockerSerivce {

    Set<DockerImage> findAllImages();

    DockerVersion getDockerVersion();

    String listProccesses(String id);

    String getImageDetails(String id);

    String getContainerDetails(String id);

    List<DockerContainer>
      findAllContainers(boolean isRunning);

    int startContainer(String id);

    int stopContainer(String id);

    int restartContainer(String id);

    int startTorBrowser();

    String getDockerHostHostName();
}
```

Die dazugehörige Implementierung benutzt keinen der vorhanden Docker Java Libraries. Stattdessen erfolgen die HTTPS Aufrufe über die Apache Commons HTTP library. [24]

Der Output wird mit GSON geparsed und angezeigt. [25]
Der Folgende Codeabschnitt zeigt exemplarisch die Implementierung um alle Docker Images abzufragen.

Das JSON Ergebnis wird anschließend in Objekte der Klasse DockerImage serialisiert und ausgegeben wie in Abbildung 24 zu sehen.

```
@Override
public Set<DockerImage> findAllImages() {
    try {
        return gson.fromJson(EntityUtils.toString(
            Request.Get(BASE_URL +
```

[24] https://hc.apache.org/
[25] https://github.com/google/gson

```
6                   DockerEndpoint.IMAGES.getPath())
7                       .execute().returnResponse().getEntity()),
8               new TypeToken<Set<DockerImage>>() {
9               }.getType());
10      } catch (JsonSyntaxException |
11            ParseException | IOException e) {
12          LOGGER.error("An error occured",e);
13          return Collections.emptySet();
14      }
15
16  }
17
18  public class DockerImage implements Serializable {
19
20      private static final long serialVersionUID = 1L;
21
22      private String id;
23
24      private String parendId;
25
26      private Set<String> repoTags;
27
28      private Long created;
29
30      private Long size;
31
32      private Long virtualSize;
33  }
```

5. **Integrationstest mit dem Überwachungsserver**
   Das Softwaredesign und die Umsetzung des Dashboard entstand in enger Zusammenarbeit mit dem Monitoring Daemon. Es wurden verschiedene Konfigurationen getestet und sowohl die Implementierungsdetails des Dashboard als auch des Daemon entsprechend angepasst.

4.5.7 Fazit und Ausblick

Es wurde eine Webapplikation entwickelt, mit der sich sowohl die akquirieren Daten anzeigen, als auch bei Bedarf auswerten und bewerten lassen. Der Fokus lag dabei auf Härtung und Isolation der einzelnen Komponenten mittels Docker, und der Sicherstellung der Vertraulichkeit und Integrität der Kom-

munikation. Auch wenn die funktionalen Anforderungen umgesetzt wurden, gibt es noch weitere Funktionen, die für zukünftige Versionen wünschenswert werden.

- Es war ursprünglich geplant, das Dashboard und die dazugehörigen Komponenten innerhalb von RancherOS zu deployen.

Aufgrund der folgenden technischen Probleme wurde, auf den Einsatz von RancherOS verzichtet:

- RancherOS wird aktiv weiterentwickelt und befindet sich in einer frühen Beta Phase.
- Während der Evaluierung, kam es immer wieder zu Abstürzen und Problemen, die in dem geforderten Zeitrahmen nicht ausreichend behandelt werden konnten
- Für das Deployment aller Komponenten wird das in Abschnitt 1.1.3 vorgestellte Tool docker-compose verwendet. Dieses wird aktuell von RancherOS nicht unterstützt.

Sobald RancherOS eine stabile Version erreicht, sollte es neu evaluiert werden und das deployment unter Umständen angepasst werden.

- Die Autorisierung und Authentifizierung gegenüber github.com ist funktional und auch für einfache Zwecke ausreichend, birgt aber einige Nachteile:
  - Abhängig von einem Drittanbieter
  - GitHub ist in den USA angesiedelt
  - Alternative OAuth Provider sind Facebook, Microsoft, Google und Twitter, die alle sie selben Nachteile wie GitHub bieten.

  Für eine zukünftige Iteration wäre eine zentrale Stelle, die die Authentifizierung und Autorisierung nutzt wünschenswert, bzw. selber implementiert.
- Hinzufügen einer Reportfunktion, die einmal pro Tag/Woche/Monat automatisiert einen Bericht erstellt, in dem festgelegt werden kann, welche Parameter ausgewertet werden sollen. Dieser Report könnte automatisiert über eine verschlüsselte E-Mail an berechtigte Benutzer versendet werden.

- Aus den selben Einschränkungen, wie in Abschnitt [3.7] erklärt, verwendet der Tomcat Server nicht die stärksten, verfügbaren Verschlüsselungsalgorithmen. In einer zukünftigen Version könnten diese nachinstalliert werden, und neu konfiguriert werden.

- Auch wenn mit der Verwendung von Docker die Grundlage für eine horizontale Skalierung und Replizierkareit geschaffen wurde, sind diese nicht konkret vorhanden. Bei einem Ausfall der Software ist die Überwachung nicht mehr gewährleistet.

- Es ist möglich bestehende Docker Images anzuzeigen und Container zu starten, stoppen und neu zu starten. Es wäre wünschenswert, wenn es in Zukunft auch möglich wäre, neue Container und Image zu erstellen und im allgemeinen mehr Funktionen der Docker API zu implementieren.

- Das Dashboard integriert auf einfache Art und Weise den im folgenden Abschnitt vorgestellten ELK Stack, in dem ein Link dazu angegeben ist. Eine engere Integration von ELK wäre wünschenswert. Dazu gehört auch, die Java Log Dateien des Dashboard mittels ELK auszuwerten.

5 ELK

5.1 Einleitung

Logdaten fallen auf vielen Geräten an. Oft werden diese lokal und dadurch dezentral gespeichert. Das macht das Auswerten dieser Daten umständlich. Durch die Trennung der Informationen verschiedener Quellen verringert sich auch deren Informationsgehalt im Gesamten. Viele Ereignisse lassen sich beispielsweise erst aus der Zusammenführung verschiedener Informationsquellen erkennen.
Fast jedes Programm verwendet unterschiedliche Log-Formate und die meisten davon wurden nicht mit Fokus auf Menschenlesbarkeit entwickelt.
Um die Informationen des gesamten „Sicheren Heimnetzwerk“ verfügbar zu machen, sollen die anfallenden Logdaten deshalb zentral gespeichert, menschenlesbar aufbereitet und visualisiert werden, wie es als Anwendungsfall „UC1“ (Funktionalität) festgelegt wurde.
Dafür wird in diesem Abschnitt der Programm-Stack ELK (kurz für Elasticsearch, Logstash und Kibana) evaluiert.

5.2 Architektur

Aus den einzelnen Komponenten für ELK und einem Wartungszugang über SSH ergibt sich folgender, erster Architekturentwurf.

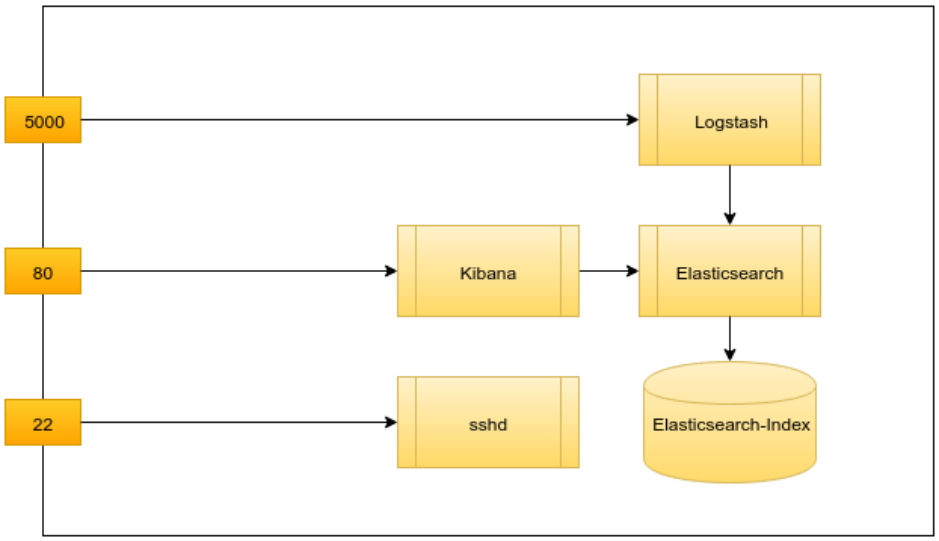


Abbildung 25: Erster Architekturentwurf

orange	Schnittstelle/Netzwerkport
gelb	Prozess/Datei

5.2.1 Sicherheitskritische Punkte der Architektur

ELK wird zwar von vielen Firmen zur Analyse im Sicherheitsumfeld genutzt, wurde aber nicht für diesen Bereich konzipiert. Da das Hauptaugenmerk bei Elasticsearch und den dafür entwickelten Anwendungen maximale Geschwindigkeit war und ist, ist die Software im Grundzustand sehr unsicher.
Folgende sicherheitskritische Punkte wurden dadurch im ersten Architekturentwurf festgestellt:

Tabelle 4: Festgestellte Probleme

Problem	**Beschreibung**
P01	Keine verschlüsselte/signierte Übertragung innerhalb ELK
P02	Keine verschlüsselte/signierte Übertragung zwischen Logstash
P03	Keine Authentifizierung in Kibana
P04	Elasticsearch und Logstash speichern unverschlüsselt

Elastic bietet mit Shield zwar eine Sicherheitslösung an, bietet diese aber nur in einer 30-tägigen Testversion und einem monatlichen „Subscription“-Modell

an[18]. Da ein großer Teil der Funktionalität von Shield nicht benötigt wird und die monatlichen Kosten von ca $90 für ein privates Projekt sehr hoch wären, muss nach einer anderen Lösung gesucht werden.

5.2.2 Lösung der Punkte

Zur Lösung der nun gesammelten Punkte werden die Maßnahmen, welche zu Beginn des Projekts in der Sicherheitskonzeption ermittelt wurden, herangezogen.

Tabelle 5: Zuvor erarbeitete Maßnahmen

Maßnahme	**Beschreibung**
M01	Verschlüsselte Datenübertragung
M02	Verschlüsselte Speicherung der Daten
M03	Signierte Datenübertragung
M04	Benutzerauthentifizierung über OAuth
(M05)	(Replizierung wichtiger Dienste)
M06	Härtung und Konfiguration der einzelnen Komponenten

Da bereits bei der Entwicklung des Dashboards gezeigt wurde, wie OAuth zu Implementieren ist und sich herausstellte, dass möglicherweise nach einer optimaleren Authentifizierungsmethode gesucht werden muss, wird an dieser Stelle auf die erneute Implementierung von OAuth verzichtet und stattdessen als vorübergehende Lösung „basic authentication“ verwendet.
Da in der Sicherheitskonzeption ermittelt wurde, dass die Verfügbarkeit der Visualisierung zweitrangig ist, wird in einer ersten Entwicklungsstufe auf Replizierung und Ausfallsicherheit verzichtet. Alle Komponenten von ELK wurden allerdings so gebaut, dass sie sich gut Replizieren und Verteilen lassen, deshalb soll dies zu einem späteren Zeitpunkt nachgeholt werden.
Zu den Problemen wurden folgende Lösungen ermittelt:

Tabelle 6: Lösungen für die festgestellten Probleme

Lösung	Beschreibung	Löst
L01	ELK in Docker, Kommunikation auf Docker begrenzen	P01
L02	Datenübertragung zwischen Logstash über stunnel	P02
L03	nginx mit SSL als Reverse Proxy vor Kibana	P03
L04	Persistierung von Elasticsearch und Logstash mit eCryptfs	P04

Daraus ergibt sich folgende neue Architektur:

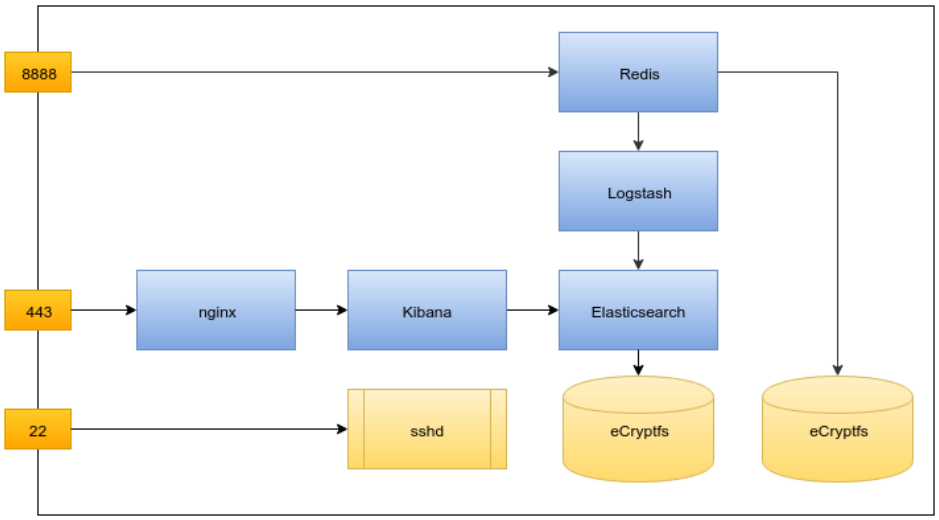


Abbildung 26: Architektur

blau	Docker-Container
orange	Schnittstelle/Netzwerkport
gelb	Prozess/Datei

Zusätzlich wird Redis verwendet, um das Schreiben mit beliebig vielen entfernten Logstash-Instanzen zu ermöglichen. Die Daten werden dort zwischengespeichert, bis sie von der lokalen Logstash-Instanz in Elasticsearch geschrieben werden.
Nachfolgend werden die Lösungen im Detail betrachtet.

1. L01: ELK in Docker, Kommunikation auf Docker begrenzen
   Die einzelnen Komponenten laufen in getrennten Docker-Containern

und sind dadurch logisch wie physikalisch voneinander und vom Host-System getrennt (separation of concerns). Von außen zugänglich gemacht werden nur der SSH-Daemon des Host-Systems, der HTTPS-Port des nginx-Proxy und der Log-Input zu Redis.

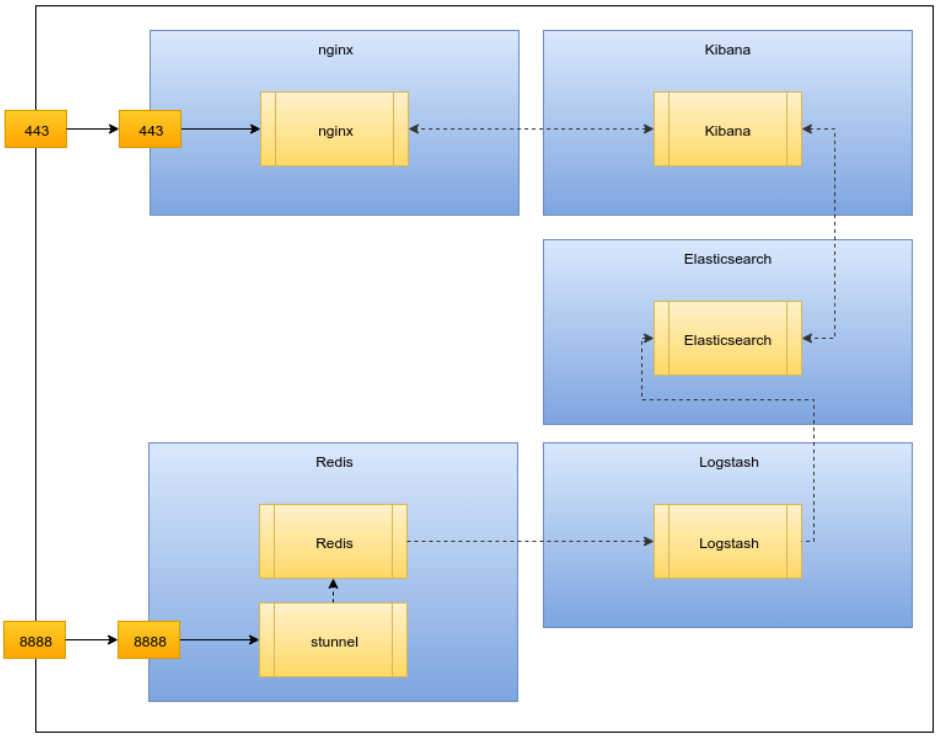


Abbildung 27: Kommunikation zwischen Docker-Containern

orange	Schnittstelle/Netzwerkport
blau	Docker-Container
gelb	Prozess
grün	SSL-Tunnel

Die Kommunikation zwischen den lokalen Docker-Containern ist zwar nicht verschlüsselt, beschränkt sich allerdings auf das Host-System und kann so auch nur mit root-Rechten mitgelesen werden. Die Ports für Logstash, Kibana und Elasticsearch werden nicht für das Host-System freigegeben (`exposed`) und sind dadurch nur für lokal laufende Docker-Container erreichbar[19, 20].

2. L02: Datenübertragung zwischen Logstash über stunnel
   Für die sichere Kommunikation zwischen entfernten mit lokal laufenden Logstash Instanzen wird stunnel verwendet. Auf den entfernten Geräten (hier beispielsweise das sichere Gateway oder die Heim-Überwachung) wird ein stunnel Client aufgesetzt, welche mit einem stunnel Server verbunden sind, über welchen sie die Daten in einer Redis-Datenbank zwischenspeichern. Aus dieser liest die lokal laufende Logstash Instanz die Daten aus und gibt sie an Elasticsearch weiter.

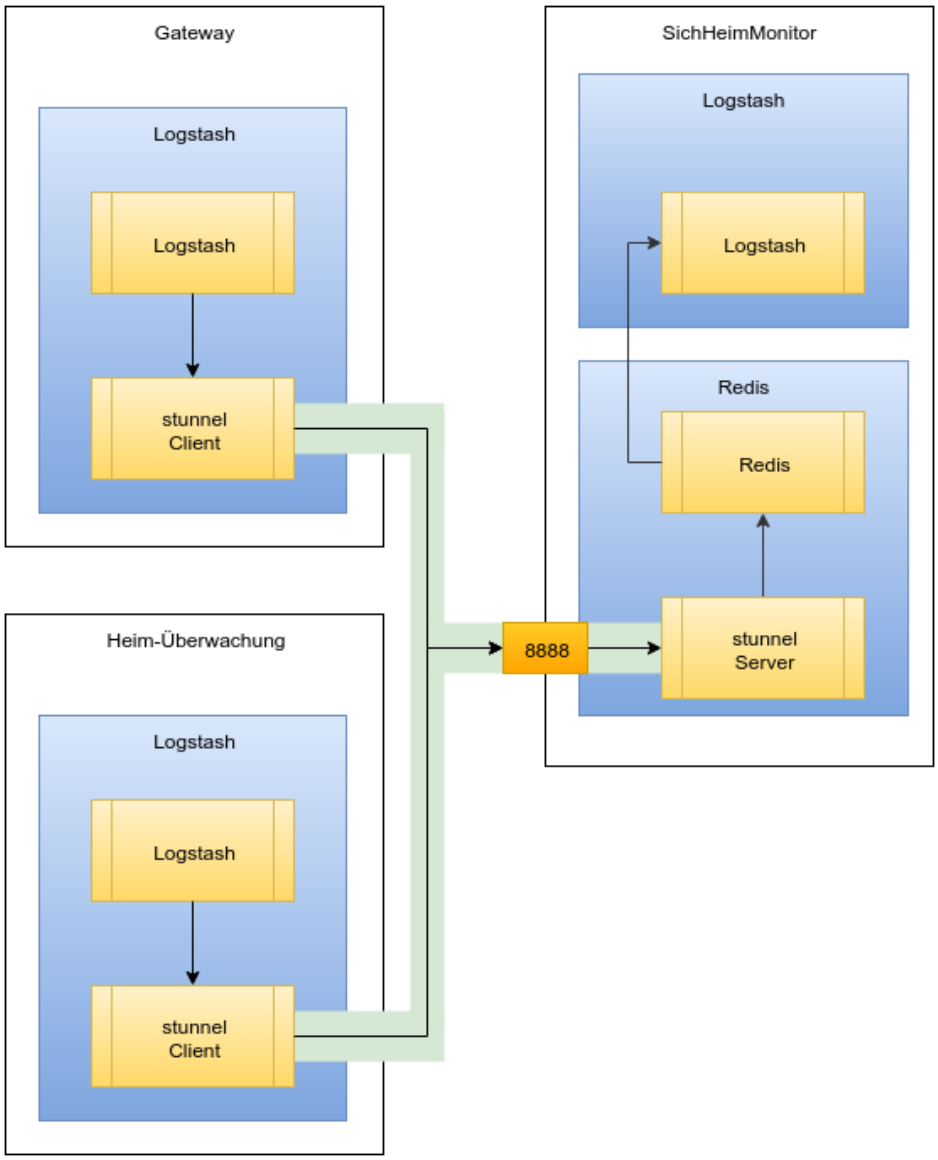


Abbildung 28: Kommunikation in Logstash

orange	Schnittstelle/Netzwerkport
blau	Docker-Container
gelb	Prozess
grün	SSL-Tunnel

Nach außen ist nur der stunnel Server erreichbar, Daten können nicht

direkt an Redis oder Logstash gesendet werden. Zusätzlich werden vom stunnel Server nur Daten angenommen, welche mit seinem eigenen public key verschlüsselt und einem ihm bekannten private key signiert sind. Umgekehrt gilt für die stunnel Clients, dass sie nur an den Server senden, wenn dieser sich mit einer korrekten Signatur ausweisen kann.

Zusätzlich wäre es auch möglich, die Hostnamen zu prüfen, und die Übertragung bei einem falschen „Common Name“ (`CN`) abzubrechen. In einem lokalen Netz sollten aber bereits die oben genannten Punkte ausreichen.

3. L03: nginx mit SSL als Reverse Proxy vor Kibana
   Um den Zugriff auf Kibana nur für berechtigte Benutzer zu ermöglichen wird nginx als Reverse-Proxy eingesetzt.

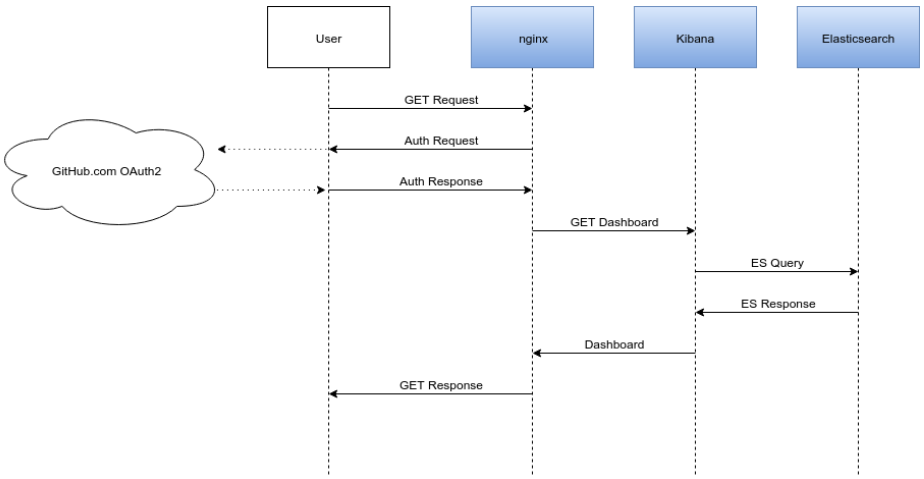


Abbildung 29: nginx als Reverse Proxy

Da jede Anfrage über nginx geht, lässt sich hier auch einfach HTTPS erzwingen und prüfen, ob der anfragende Benutzer autorisiert ist.

Da nginx von sich aus kein OAuth unterstützt, wird für einen ersten Test „basic authentication“ eingesetzt. In einer späteren Version soll dies allerdings, wie im Dashboard, durch eine Authentifikation gegen OAuth ersetzt werden.

4. L04: Persistierung von Elasticsearch und Logstash mit eCryptfs
   Um den Elasticsearch Index und den Zwischenspeicher von Logstash,

für den Redis verwendet wird, sicher zu speichern, wird mit eCryptfs ein verschlüsseltes Volumen erstellt. Dieses wird dann in den Docker-Container eingebunden, wo es wie ein normaler Ordner benutzt werden kann.

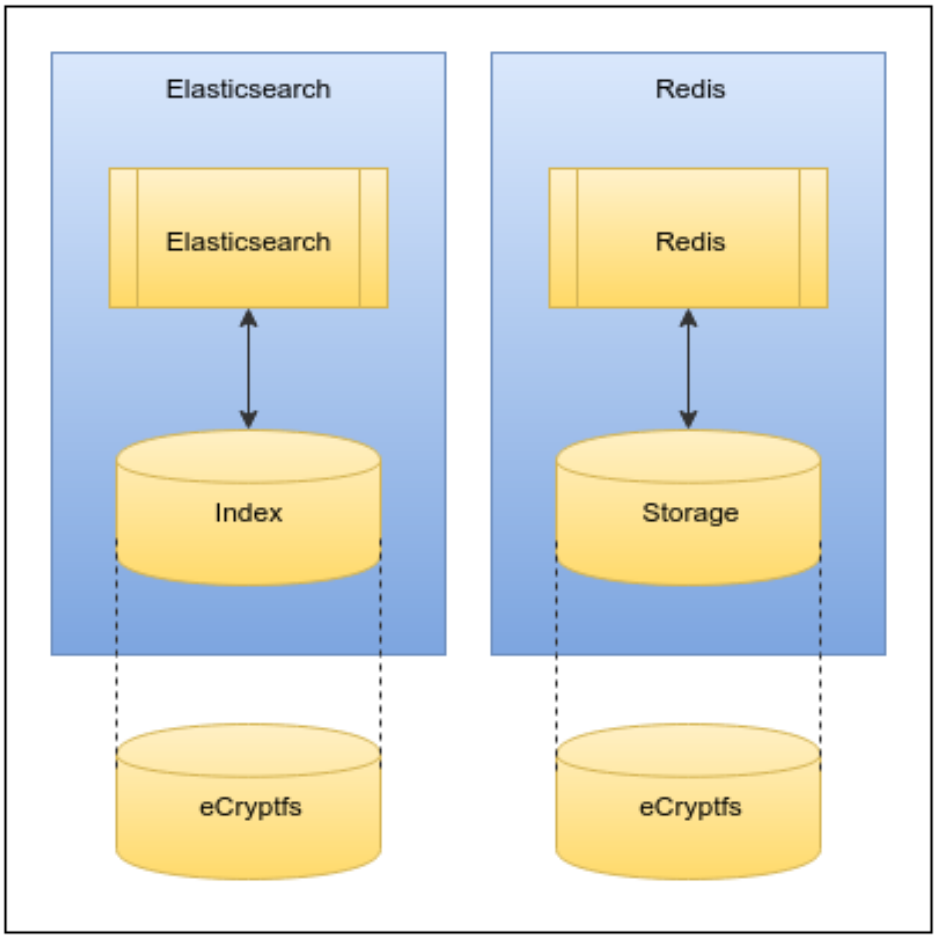


Abbildung 30: Persistierung mit eCryptfs

blau Docker-Container
gelb Prozess/Datei

5.3 Implementierung

Nachdem bereits für das „SichHeimMonitor Dashboard“ gezeigt wurde, wie man ein Docker-Image von Grund auf sicher baut, wird hier für einen ers-

ten Implementierungs-Prototypen darauf verzichtet und die auf Docker-Hub bereitgestellten Images als Grundlage verwendet.
Ein Docker-Image, welches speziell mit Fokus auf Sicherheit gebaut wurde, ist einem sehr allgemein gehaltenen Image zwar in jedem Fall vorzuziehen, durch die Veröffentlichung von Dockerfiles auf Docker-Hub kann aber zumindest der Entstehungsprozess des Images nachverfolgt werden.
Da Ordner und Dateien im Docker-Container beliebig eingehängt werden können, ist für die Implementierung keine feste Ordnerstruktur notwendig. Der Einfachheit halber werden aber alle Dateien zu einem Docker-Container in einem Ordner gespeichert.

5.3.1 Konfiguration von nginx

1. HTTP Authentication
   Zur Erzeugung und Verwendung der Authentifikationsdaten für nginx wird das sehr ausführliche Tutorial von digitalocean[21] verwendet.

   Mit dem Programm `htpasswd` aus den Apache Utils werden Logindaten erzeugt und in der Datei `./htpasswd` gespeichert. Auf diese Datei wird später in nginx referenziert.

```
1 htpasswd -c ./htpasswd benutzer1
2 htpasswd ./htpasswd benutzer2
```

2. Server Zertifikat
   Da der Server nur privat genutzt wird und die Zertifizierung im besten Fall nur gegenüber des „SichHeimMonitor Dashboards“ stattfindet, reicht hier ein selbst signiertes Zertifikat.

   Auch hierfür gibt es wieder ein sehr ausführliches Tutorial von digitalocean[22].

   Mit `openssl` werden Schlüssel und Zertifikat erzeugt und im Ordner `$nxdir` gespeichert. Auch diese Dateien werden später in nginx referenziert.

```
1 openssl req -x509 -nodes -days 365 \
2   -newkey rsa:2048 \
3   -subj '/CN=nginx/O=SSH/C=DE' \
4   -keyout $nxdir/ssl.key \
5   -out $nxdir/ssl.crt
```

3. Besonderheiten für Docker
   Bei der Konfiguration von nginx unter Docker ist zu beachten, dass man nginx mit `worker_processes 1;` auf einen Prozess begrenzt und mit `daemon off;` der Prozess im Vordergrund gehalten wird.

   Ansonsten ist die Konfiguration relativ simpel und wird wie in diesem Tutorial[23] beschrieben durchgeführt.

4. Sichere SSL-Konfiguration
   Zur Härtung von nginx wird der Server wie in diesen Artikeln[24, 25] konfiguriert. Neben den beschriebenen Einstellungen für HTTP Strict Transport Security und Online Certificat Status Protocol (OCSP) Stapling, werden zusätzlich die Protokolle „TLS 1.0“ und „TLS 1.1“ deaktiviert wie auch ein geringerer Satz an Cipher Suites gewählt. Dies führt zwar dazu, dass ältere Browser nicht mehr unterstützt werden, da der Dienst aber nur von uns selbst genutzt werden soll, ist das hier kein Problem. Zusätzlich werden für das Diffie-Hellman-Verfahren zum Schlüsselaustasch neue Parameter erzeugt. Dies ist deshalb wichtig, da sonst die mitgelieferten und für alle Installationen gleichen Parameter verwendet werden.

5. Ausblenden von Informationen in Fehlerseiten
   Im Auslieferungszustand werden in Fehlerseiten von nginx der verwendete Webserver-Name sowie die Versionsnummer ausgegeben. Dadurch würde es bei aufkommenden Sicherheitslücken Angreifern erleichtert werden, diese auszunutzen. Allgemein wird deshalb dazu geraten, die Versionsnummer auszublenden oder die Fehlerseiten komplett zu ändern[26]. Um diese Ausnutzung zu verhindern, soll bei auftretenden Fehlern eine leere Seite ausgeliefert werden. Dies ist zwar für Endanwender im Fehlerfall nicht sehr informativ, da der Dienst aber ausschließlich von uns selbst verwendet wird, ist dies nicht relevant, da wir auch Zugriff auf die Fehler-Logs von nginx haben.

6. Abschließende Konfigurationsdatei

Listing 10: nginx.conf

```
1 user  nginx;
2 worker_processes  1;
3 ...
```

```
4
5  http {
6    add_header Strict-Transport-Security "max-age=31536000; \
7  ␣␣␣␣includeSubdomains; preload";
8    add_header X-Frame-Options DENY;
9    add_header X-Content-Type-Options nosniff;
10
11   ssl_protocols TLSv1.2;
12
13   ssl_ciphers AES256+EECDH:AES256+EDH:!aNULL;
14   ssl_ecdh_curve secp384r1;
15
16   ssl_stapling on;
17   ssl_stapling_verify on;
18
19   ssl_certificate         /var/ssl-cert/ssl.crt;
20   ssl_certificate_key     /var/ssl-cert/ssl.key;
21   ssl_trusted_certificate /var/ssl-cert/ssl.crt;
22   ssl_session_timeout 10m;
23   ssl_session_cache shared:SSL:10m;
24   ssl_session_tickets off;
25   ssl_dhparam             /etc/ssl/certs/dhparam.pem;
26
27   server {
28     listen 443 ssl;
29
30     auth_basic              "Restricted";
31     auth_basic_user_file    /etc/nginx/.htpasswd;
32
33     error_pages 400 401 ... /error/err.html;
34
35     location / {
36       proxy_pass http://kibana:5601;
37       proxy_http_version 1.1;
38       proxy_set_header Upgrade $http_upgrade;
39       proxy_set_header Connection 'upgrade';
40       proxy_set_header Host $host;
41       proxy_cache_bypass $http_upgrade;
42     }
43
44     location ^~ /error/ {
45       alias /usr/share/nginx/html/;
46       internal;
47     }
48   }
```

```
49    ...
50  }
```

7. Erzeugen des Docker-Images
   Das Docker-Image wird mit dem Befehl

```
1  docker build -t elk-nginx ./nginx
```

gebaut. `./nginx` ist dabei der Ordner in dem sich die Dateien `nginx.conf`, `htpasswd` und der erzeugte SSL-Schlüssel befinden. Auch ist dort das nachfolgende `Dockerfile` enthalten, mit welchem das Image erzeugt wird.

Listing 11: nginx Dockerfile

```
1  FROM nginx:latest
2  MAINTAINER Armin Grodon <me@armingrodon.de>
3
4  RUN openssl dhparam -dsaparam \
5    -out /etc/ssl/certs/dhparam.pem 2048
6
7  VOLUME ["/var/cache/nginx"]
8
9  EXPOSE 80 443
10
11 CMD ["nginx", "-g", "daemon␣off;"]
```

Mit dem Befehl `openssl dhparam` werden beim Erstellen des Images neue Parameter für das Diffie-Hellman-Verfahren erzeugt[25]. Hierbei wird während der Testphase die Parameter mit der Option `-dsaparam` als „DSA Parameter“ erzeugt und in „Diffie-Hellman Parameter“ umgewandelt, da das Erzeugen von 4096 bit „Diffie-Hellman Parameter“ zwischen einigen Stunden und wenigen Tagen dauern kann. Im Produktiveinsatz sollte die Zeile aber durch

```
1  RUN openssl dhparam -out /etc/ssl/certs/dhparam.pem 4096
```

ersetzt werden.

Mit `EXPOSE` wird angegeben, dass es spät möglich ist, für den Docker-Container die Ports 80 und 443 frei zu geben.

5.3.2 Konfiguration von Logstash

Die Konfiguration von Logstash unterteilt sich in den Server, welcher die Daten aus Redis liest und in Elasticsearch speichert, dem Zwischenspeicher, für welchen Redis verwendet wird und der Client, welcher die Daten an Redis sendet.

1. Grok-Parser für SSH und iptables
   Um das Log-Format von `iptables` richtig interpretieren zu können, benötigt Logstash zusätzliche Informationen. Diese können beispielsweise in einer Art regulärem Ausdruck als „grok"-Filter angegeben werden[27]. Diese bestehen aus Text, der identisch sein muss und Platzhaltern, welche gegebenenfalls einem bestimmten Typen entsprechen müssen. Die Werte die den Platzhaltern entsprechen, werden dann mit spezifiziertem Typen an Elasticsearch weitergegeben.

   Einfachere Regeln können direkt in die Logstash-Konfiguration geschrieben werden.

   Listing 12: Grok-Filter für SSH

```
1 grok {
2   match => [ "message",
3     "%{SYSLOGTIMESTAMP:timestamp}␣%{HOSTNAME:host_target}␣\
4 ␣␣␣␣␣␣sshd\[%{BASE10NUM}\]:␣Failed␣password␣for␣invalid␣user␣\
5 ␣␣␣␣␣␣%{USERNAME:username}␣from␣%{IP:src_ip}␣port␣\
6 ␣␣␣␣␣␣%{BASE10NUM:port}␣ssh2" ]
7   add_tag => [ "ssh_brute_force_attack" ]
8 }
```

   Dieser Filter trifft zum Beispiel auf alle SSH-Logs zu, welche durch die Verwendung eines falschen Benutzernamen und Passworts erzeugt werden. Neben den Angaben wie Zeitpunkt, Benutzername und IP wird diese Anfrage mit einem zusätzlichen „Tag" gekennzeichnet, in diesem Fall um die Anfrage als vermutlichen Brute-Force-Angriff zu kennzeichnen. Dadurch kann später besser nach bestimmten Ereignissen gesucht werden.

   Für kompliziertere Filter können eigene Dateien angelegt werden. Diese enthalten für jede neue Zeile ein Paar aus Alias und Filter. Dadurch kann man verschachtelte Filter schreiben und den Alias in der Konfiguration verwenden.

Listing 13: Grok-Filter für iptables

```
NETFILTERMAC %{COMMONMAC:dst_mac}:%{COMMONMAC:src_mac}:\
  %{ETHTYPE:ethtype}
ETHTYPE (?:(?:[A-Fa-f0-9]{2}):(?:[A-Fa-f0-9]{2}))
IPTABLES1 (?:IN=%{WORD:in_device} OUT=(%{WORD:out_device})?
  MAC=%{NETFILTERMAC} SRC=%{IP:src_ip}
  DST=%{IP:dst_ip}.*(TTL=%{INT:ttl})?.*PROTO=%{WORD:proto}?\
  .*SPT=%{INT:src_port}?.*DPT=%{INT:dst_port}?.*)
IPTABLES2 (?:IN=%{WORD:in_device} OUT=(%{WORD:out_device})?
  MAC=%{NETFILTERMAC} SRC=%{IP:src_ip}
  DST=%{IP:dst_ip}.*(TTL=%{INT:ttl})?.*PROTO=%{INT:proto}?.*)
IPTABLES (?:%{IPTABLES1}|%{IPTABLES2})
```

Da die einzelnen Filter zu lang für eine Zeile waren, wurden sie hier umgebrochen.

Der letzte Filter `IPTABLES`, welcher wiederum entweder dem Filter `IPTABLES1` oder `IPTABLES2` entsprechen muss, kann dann in der Konfiguration verwendet werden.

```
grok {
  match => [ "message", "%{IPTABLES}" ]
  patterns_dir => [ "/var/lib/logstash/grok" ]
}
```

2. Konfiguration des Servers
   Möchte man, dass die lokal laufende Logstash-Instanz neben den zwischengespeicherten Daten aus Redis auch die lokalen Daten aus SSH (`auth.log`) und iptables (`kern.log`) an Elasticsearch sendet, sieht die Konfiguration folgendermaßen aus:

Listing 14: logstash.conf des Servers

```
input {
  file {
    type => "linux-syslog"
    path => "/var/host/auth.log"
  }

  file {
    type => "kern-log"
    path => "/var/host/kern.log"
  }

```

```
12    redis {
13      host => "redis"
14      data_type => "list"
15      key => "logstash"
16      codec => json
17    }
18  }
19
20  filter {
21    ...
22  }
23
24  output {
25    elasticsearch {
26      action => "index"
27      hosts => "elasticsearch:9200"
28      index => "logstash-host"
29      workers => 1
30    }
31  }
```

3. Erzeugen des Docker-Images für den Server
   Das Docker-Image wird mit dem Befehl

```
1  docker build -t elk-logstash ./logstash
```

   gebaut. `./logstash` ist dabei der Ordner in dem sich der Grok-Filter `iptables` und und die Konfiguration befindet. Auch ist dort das nachfolgende `Dockerfile` enthalten, mit welchem das Image erzeugt wird.

Listing 15: Logstash Dockerfile

```
1  FROM logstash:latest
2  MAINTAINER Armin Grodon <me@armingrodon.de>
3
4  RUN usermod -G adm logstash
5
6  RUN mkdir -p /var/lib/logstash/grok
7  COPY iptables /var/lib/logstash/grok
8
9  EXPOSE 5000
10
11 ENTRYPOINT ["/docker-entrypoint.sh"]
12 CMD ["logstash", "-f", "/etc/logstash/conf.d/logstash.conf"]
```

Um Leserechte für die Logdateien zu erhalten, wird der Benutzer `logstash` in die Gruppe `adm` aufgenommen.

4. Konfiguration von Redis
   Für Redis selbst muss keine Konfiguration vorgenommen werden, allerdings für den stunnel Server, welcher mit Redis in einem Container läuft.

   Auf Client- und Server-Seite werden RSA-Schlüssel und -Zertifikate erstellt, die Zertifikate werden ausgetauscht. Die Daten des Servers werden in `$stunnels` und die des Clients in `$stunnelc` gespeichert. Diese Ordner werden dann später in das Docker-Image eingebunden.

Listing 16: Erzeugung der Schlüssel für stunnel

```
# client (shipper) stunnel
openssl genrsa \
    -out $stunnelc/client.key 2048
openssl req -new -x509 -nodes -days 365 \
    -subj '/CN=logstash/O=SHM/C=DE' \
    -key $stunnelc/client.key \
    -out $stunnelc/client.crt
cp $stunnelc/client.crt $stunnels

# server (redis) stunnel
openssl genrsa \
    -out $stunnels/server.key 2048
openssl req -new -x509 -nodes -days 365 \
    -subj '/CN=redis/O=SHM/C=DE' \
    -key $stunnels/server.key \
    -out $stunnels/server.crt
cp $stunnels/server.crt $stunnelc
```

Zusätzlich wird ein Schlüssel für eCryptfs aus Zufallsdaten erstellt und an einem sicheren Ort, beispielsweise einem verschlüsselten home-Ordner oder einem verschlüsselten, externen Datenträger, als `$rekey` gespeichert.

```
echo "passwd=" > $rekey
sudo dd if=/dev/urandom count=1 bs=32 >> $rekey
```

In der stunnel-Konfiguration wird als Eingangsport `8888` und als Ausgangsport `6379` verwendet. Dadurch werden alle eingehenden Daten an Redis weitergeleitet.

Listing 17: stunnel.conf des Servers

```
verify = 2
client = no
pid = /var/run/stunnel.pid

[logstash]
accept = 8888
connect = 127.0.0.1:6379
cert = /etc/stunnel/server.crt
key = /etc/stunnel/server.key
CAfile = /etc/stunnel/client.crt
```

Mit `verify = 2` wird angegeben, dass auf Verschlüsselung und Signatur geachtet wird, nicht aber auf den Hostnamen.

5. Erzeugen des Docker-Images für Redis Das Docker-Image wird mit dem Befehl

```
docker build -t elk-redis ./redis
```

gebaut. `./redis` ist dabei der Ordner in dem sich der Schlüssel und das Zertifikat des Servers, sowie die Zertifikate aller Clients befinden. Auch ist dort das nachfolgende `Dockerfile` enthalten, mit welchem das Image erzeugt wird.

Listing 18: Redis Dockerfile

```
FROM redis:latest

RUN apt-get update && apt-get install \
  -y --no-install-recommends \
  stunnel4 \
  && rm -rf /var/lib/apt/lists/*

RUN sed -i "s/ENABLED=0/ENABLED=1/" /etc/default/stunnel4

COPY entrypoint.sh /

EXPOSE 8888

ENTRYPOINT [ "/entrypoint.sh" ]
CMD [ "redis-server" ]
```

Im vorgefertigten Docker-Image (`redis:latest`) wird stunnel installiert und mit dem `sed`-Befehl aktiviert.

6. Konfiguration des Clients
   Soll eine entfernte Logstash-Instanz ein Datei, hier `/var/input.txt` überwachen, sieht die Konfiguration folgendermaßen aus:

Listing 19: logstash.conf des Clients

```
input {
  file {
    path => "/var/input.txt"
  }
}

output {
  redis {
    host => "localhost"
    data_type => "list"
    key => "logstash-remote"
    codec => json
  }
}
```

   Dadurch werden die Daten an den stunnel Client gesendet, welchen Logstash für eine Redis Datenbank hält.

   Stunnel wird ähnlich wie in Abschnitt 4 konfiguriert, das verwendete Schlüsselpaar wurde bereits in diesem Abschnitt erzeugt.

Listing 20: stunnel.conf des Clients

```
verify = 2
client = yes
pid = /var/run/stunnel.pid

[logstash]
accept = 6379
connect = redis:8888
cert = /etc/stunnel/client.crt
key = /etc/stunnel/client.key
CAfile = /etc/stunnel/server.crt
```

   Im Gegensatz zum Server gibt sich der Client allerdings als lokal laufende Redis-Instanz aus. Der Name `redis` muss in `/etc/hosts` auf die Adresse des Hosts zeigen, auf dem ELK läuft.

7. Erzeugen des Docker-Images für den Client
   Das Docker-Image wird mit dem Befehl

```
docker build -t elk-shipper ./shipper
```

gebaut. `./shipper` ist dabei der Ordner in dem sich eventuelle Filter, die Konfiguration und alle Dateien für den stunnel Client befindet. Auch ist dort das nachfolgende `Dockerfile` enthalten, mit welchem das Image erzeugt wird.

Listing 21: Shipper Dockerfile

```
FROM logstash:latest
MAINTAINER Armin Grodon <me@armingrodon.de>

RUN apt-get update && apt-get install \
  -y --no-install-recommends \
  stunnel4 \
  && rm -rf /var/lib/apt/lists/*

RUN sed -i "s/ENABLED=0/ENABLED=1/" /etc/default/stunnel4

COPY entrypoint.sh /
COPY stunnel/* /etc/stunnel/
COPY logstash.conf /etc/logstash/conf.d/logstash.conf

ENTRYPOINT [ "/entrypoint.sh" ]
CMD [ "logstash", "-f", "/etc/logstash/conf.d/logstash.conf" ]
```

Um das Image verteilen zu können, wird es exportiert

```
# export shipper image (local host)
docker save elk-shipper > shipper.tar
# import shipper image (remote host)
docker load < shipper.tar
```

5.3.3 Konfiguration von Elasticsearch

Da die Speicherung von Daten aus Logstash und das Anzeigen dieser in Kibana ein Standard-Anwendungsfall von Elasticsearch ist, muss an Elasticsearch selbst keine Konfiguration vorgenommen werden.
Es muss lediglich der Schlüssel für eCryptfs erzeugt werden.

1. Erzeugen des eCryptfs-Schlüssels
   Ähnlich wie in 4 wird ein Schlüssel aus Zufallsdaten erzeugt und in

`$eskey` gespeichert. Für diesen Schlüssel gelten ebenfalls die gleichen Sicherheitsanforderungen.

```
1 echo "passwd=" > $eskey
2 sudo dd if=/dev/urandom count=1 bs=32 >> $eskey
```

2. Erzeugen des Docker-Images
   Das Docker-Image wird mit dem Befehl

```
1 docker build -t elk-elasticsearch ./elasticsearch
```

gebaut. `./elasticsearch` ist dabei der Ordner, der das nachfolgende `Dockerfile` enthält, mit welchem das Image erzeugt wird.

Listing 22: Elasticsearch Dockerfile

```
1 FROM elasticsearch:latest
2 MAINTAINER Armin Grodon <me@armingrodon.de>
3
4 CMD [ "elasticsearch", "-Des.network.host=0.0.0.0" ]
```

5.3.4 Konfiguration von Kibana

Bevor Kibana gestartet wird muss Elasticsearch bereits laufen. Dafür wird ein kleines Skript erstellt, in welchem mit `netcat` geprüft wird, ob eine Verbindung zu Elasticsearch aufgebaut werden kann.

Listing 23: waitES.sh

```
1 while true; do
2     nc -q 1 elasticsearch 9200 \
3     2> /dev/null && break
4 done
5
6 kibana
```

1. kibana.yml
   In der Konfigurationsdatei von Kibana wird festgelegt, auf welchem Port gelauscht werden soll, auf welchem Host und Port Elasticsearch läuft und welche Plugins verwendet werden sollen.

Listing 24: kibana.yml

```
port: 5601
host: "0.0.0.0"

elasticsearch_url: "http://elasticsearch:9200"
elasticsearch_preserve_host: true

kibana_index: ".kibana"

default_app_id: "discover"
request_timeout: 300000
shard_timeout: 0

bundled_plugin_ids:
    - plugins/dashboard/index
    - plugins/discover/index
    - plugins/doc/index
    - plugins/kibana/index
    - plugins/markdown_vis/index
    - plugins/metric_vis/index
    - plugins/settings/index
    - plugins/table_vis/index
    - plugins/vis_types/index
    - plugins/visualize/index
```

2. Erzeugen des Docker-Images
   Das Docker-Image wird mit dem Befehl

```
docker build -t elk-kibana ./kibana
```

gebaut. `./kibana` ist dabei der Ordner in dem sich die Dateien `kibana.yml` und das Skript `waitES.sh` befindet. Auch ist dort das nachfolgende `Dockerfile` enthalten, mit welchem das Image erzeugt wird.

Listing 25: Kibana Dockerfile

```
FROM kibana:latest
MAINTAINER Armin Grodon <me@armingrodon.de>

RUN apt-get update
RUN apt-get -y --force-yes install netcat

COPY waitES.sh /tmp/waitES.sh
```

```
8  RUN chmod +x /tmp/waitES.sh
9
10 RUN kibana plugin --install elastic/sense
11
12 EXPOSE 5601
13
14 CMD ["/tmp/waitES.sh"]
```

3. Einrichten eines Dashboards
   Der Hauptteil der Konfiguration von Kibana findet nach dem Start statt.

   Ziel ist es ein Dashboard mit grafischen Darstellungen der gesammelten Daten zu erzeugen.

   Nachdem genug Daten in Elasticsearch gesammelt sind, muss in Kibana ein „index pattern“ ausgewählt werden um in Elasticsearch zu suchen.

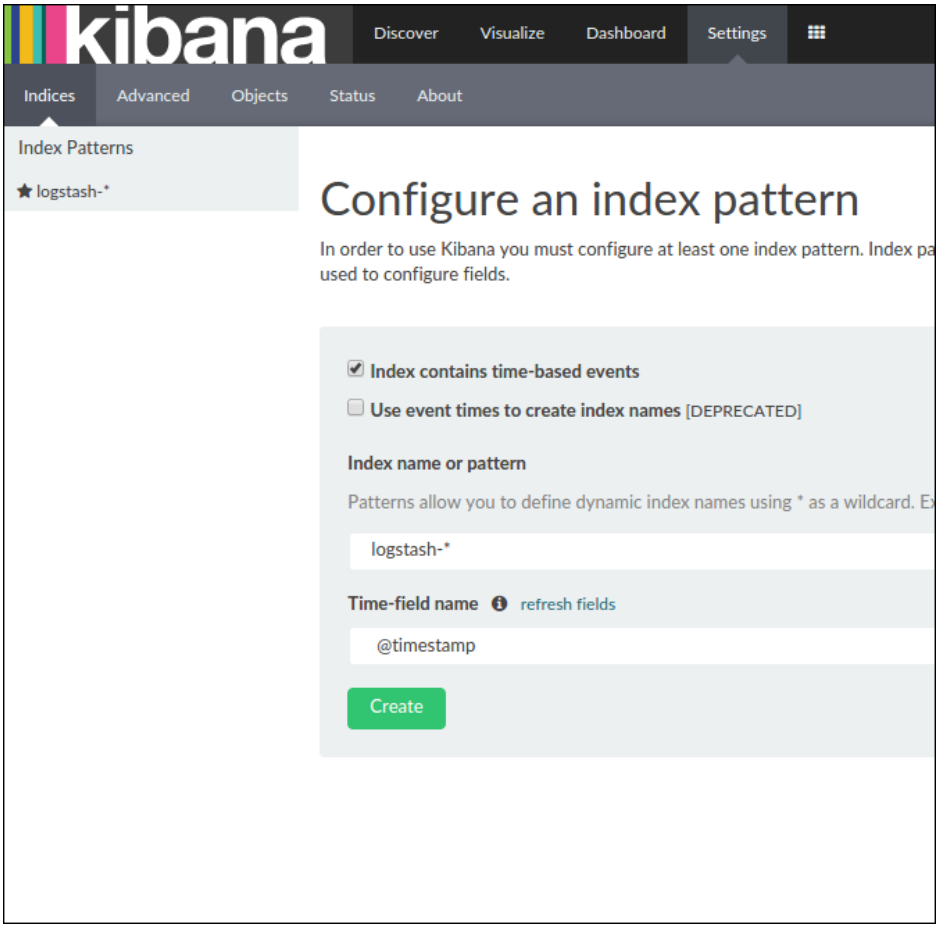


Abbildung 31: Erzeugung eines index pattern

Anschließend lassen sich im Reiter „Visualize“ Diagramme aus Suchanfragen erzeugen. Zum Beispiel ein Kreisdiagramm aus allen Netzwerkanfragen (hier ohne Broadcasts), aufgeteilt nach den 10 am Meisten angefragten Ports.

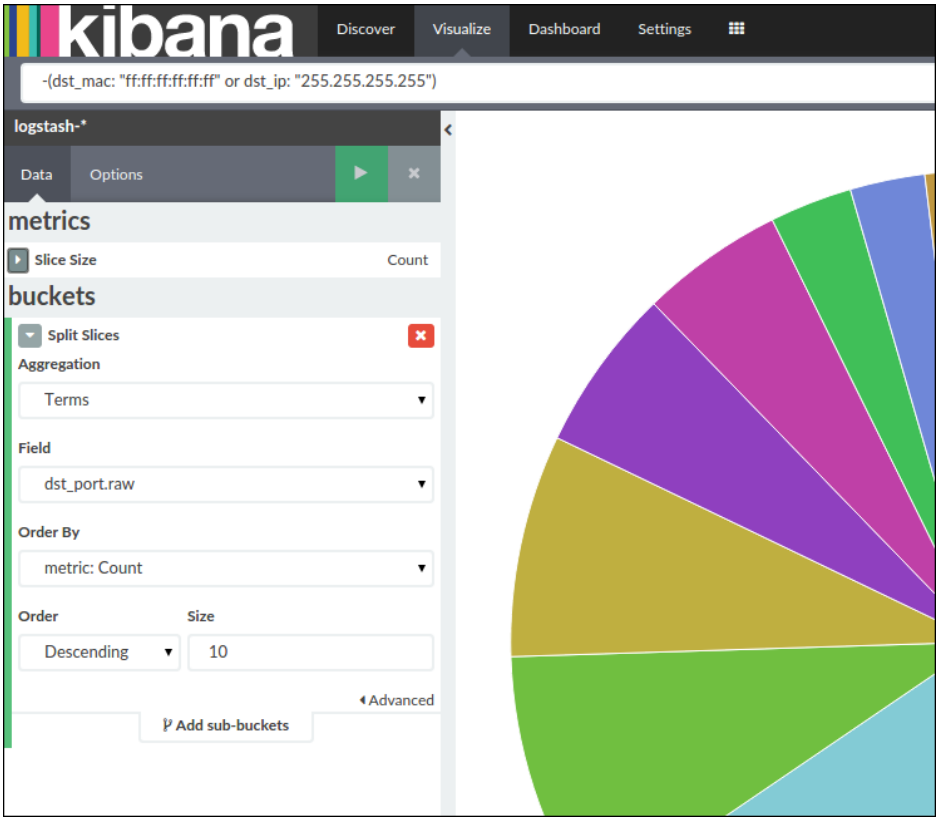


Abbildung 32: Erstellung eines Diagramms

Hat man genug Diagramme erstellt, lassen sich diese in zusammen in einem Dashboard speichern.

Abbildung 33: Kibana Dashboard

5.3.5 Zusammenführung in Skripten

Die Befehle der obigen Abschnitte werden in drei Skripten zusammengefasst. `make.sh` enthält die Erzeugung aller Schlüssel und die Generierung der Docker-Images. Es wird einmal, am Anfang, zur Initialisierung ausgeführt.
`start.sh` enthält das mounten der eCryptfs-Volumen und das Starten und Verknüpfen aller lokalen Docker-Container.
`stop.sh` stoppt alle Docker-Container und verschlüsselt die eCryptfs-Volumen wieder.

5.3.6 Sonstige Aufgaben

Neben der Konfiguration der Komponenten von ELK fallen Aufgaben zur Absicherung an, auf die hier nicht im Detail eingegangen wird.

1. Härtung des Servers
   Der SSH Zugang wird nach dem OWASP Guideline für Web Application Server[28] gehärtet und iptables nach selbigem Leitfaden konfiguriert, um eingehenden Verkehr nur noch für die eingehend Ports aus Kapitel 5.2.2 zuzulassen.

5.4 Test und Verifizierung

Um sicher zu gehen, dass die in Abschnitt 5.2.2 genannten Maßnahmen eingehalten werden, muss die Implementierung getestet werden.

5.4.1 L01: Kommunikation zwischen Docker-Containern

Die getroffene Annahme ist, dass auf Docker-Container, welche keine Ports freigegeben haben (z.B. mit =-p "80:80-) nicht vom Hostsystem zugegriffen werden kann.
Um dies zu überprüfen, starten wir zunächst einen Docker-Container mit:

```
1 docker run -it --rm -p "80:80" nginx:latest
```

Dieser Container verwendet intern die Ports 80 un 443, von denen wir explizit den Port 80 an das Host-System freigeben. Fragen wir nun diesen Port am virtuellen Docker-Host (172.17.0.1) an, erhalten wir als Ausgabe die Startseite von nginx.

```
1 $ curl 172.17.0.1:80
2 <!DOCTYPE html>
3 ...
```

Starten wir nun aber selbigen Container ohne den Port frei zu geben,

```
1 docker run -it --rm nginx:latest
```

erhalten wir auf die gleiche Anfrage keine Antwort.

```
1 $ curl 172.17.0.1:80
2 curl: (7) Failed to connect ... Connection refused
```

5.4.2 L02: Datenübertragung zwischen Logstash

Hier ist die getroffene Annahme, dass die Kommunikation zwischen dem Logstash-Shipper und dem lokal laufenden Logstash durch stunnel verschlüsselt und signiert stattfindet. Da wir möglichst andere Faktoren und Nachrichten ausschließen wollen und der essenzielle Part die Kommunikation über stunnel ist, wird als Testaufbau wieder nginx und der Einfachheit halber ein Ubuntu-Container gewählt, dabei wird ähnlich wie in der Konfiguration von Logstash vorgegangen. Den nginx-Container starten wir mit:

```
1 docker run -it --rm -p "80:80" -p "8888:8888" nginx:latest
```

stunnel wird auf dem Container so konfiguriert, dass es auf Port 8888 lauscht und auf Port 80 weiterleitet. Dadurch sollte der Client auf Port 80 unverschlüsselt und auf Port 8888, über stunnel, verschlüsselt mit dem Webserver kommunizieren können.
Auf dem Ubuntu-Container wird ebenfalls stunnel installiert und so konfiguriert, dass es auf dem lokalen Port 80 lauscht und an Port 8888 des Host-Systems weiterleitet, auf welchem der nginx-Container lauscht.
Abschließend kopieren wir die für Redis und Shipper generierten Schlüssel auf die Container.

1. Verschlüsselung
   Um zu überprüfen, ob die Daten verschlüsselt übertragen werden, werden wir einmal eine unverschlüsselte und anschließend eine verschlüsselte Verbindung zwischen Client (Ubuntu) und Server (nginx) aufbauen und beide Verbindungen mit Wireshark aufzeichnen.

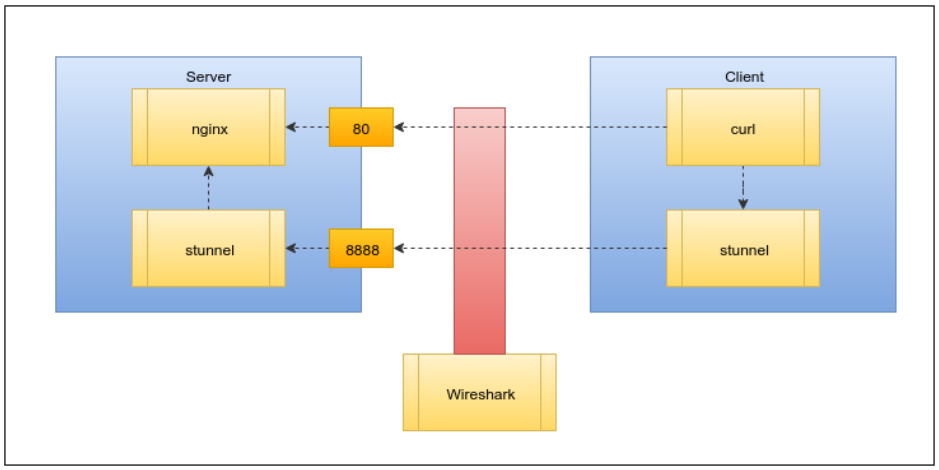


Abbildung 34: Wireshark Versuchsaufbau

blau	Docker-Container
orange	Schnittstelle/Netzwerkport
gelb	Prozess
rot	Wireshark im Netzwerk

Baut der Client die Verbindung direkt zu Port 80 des Servers auf, sehen wir in Wireshark folgende Ausgabe:

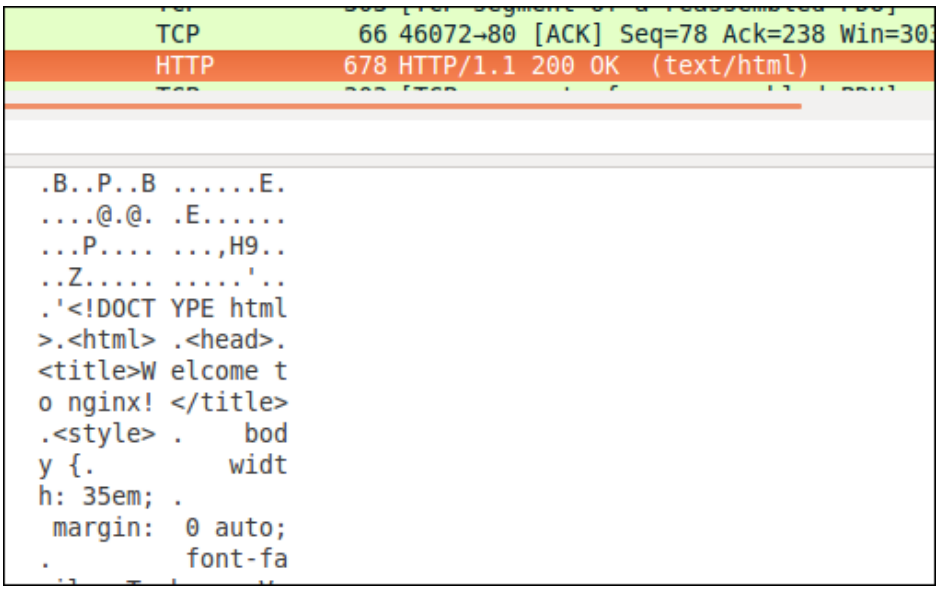
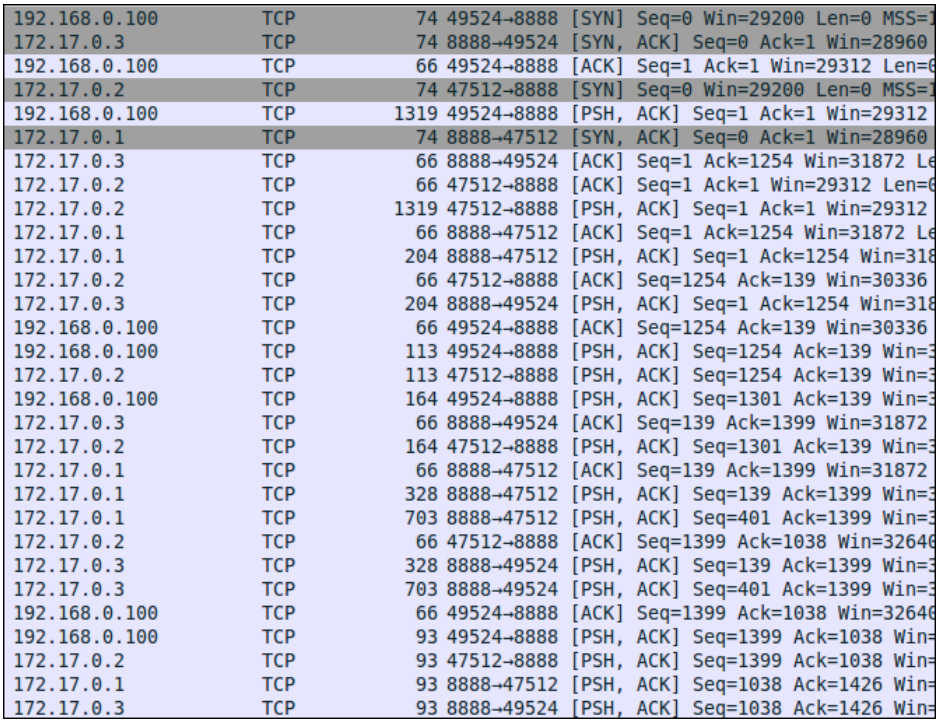


Abbildung 35: Klartext-Mitschnitt von Wireshark

Wie zu erwarten ist die Übertragung in Klartext. Baut der Client die Verbindung stattdessen über stunnel auf, sehen wir in Wireshark weder übertragene Protokolle, noch den darin enthaltenen Inhalt.



192.168.0.100	TCP	74	49524→8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1
172.17.0.3	TCP	74	8888→49524 [SYN, ACK] Seq=0 Ack=1 Win=28960
192.168.0.100	TCP	66	49524→8888 [ACK] Seq=1 Ack=1 Win=29312 Len=0
172.17.0.2	TCP	74	47512→8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1
192.168.0.100	TCP	1319	49524→8888 [PSH, ACK] Seq=1 Ack=1 Win=29312
172.17.0.1	TCP	74	8888→47512 [SYN, ACK] Seq=0 Ack=1 Win=28960
172.17.0.3	TCP	66	8888→49524 [ACK] Seq=1 Ack=1254 Win=31872 Le
172.17.0.2	TCP	66	47512→8888 [ACK] Seq=1 Ack=1 Win=29312 Len=0
172.17.0.2	TCP	1319	47512→8888 [PSH, ACK] Seq=1 Ack=1 Win=29312
172.17.0.1	TCP	66	8888→47512 [ACK] Seq=1 Ack=1254 Win=31872 Le
172.17.0.1	TCP	204	8888→47512 [PSH, ACK] Seq=1 Ack=1254 Win=318
172.17.0.2	TCP	66	47512→8888 [ACK] Seq=1254 Ack=139 Win=30336
172.17.0.3	TCP	204	8888→49524 [PSH, ACK] Seq=1 Ack=1254 Win=318
192.168.0.100	TCP	66	49524→8888 [ACK] Seq=1254 Ack=139 Win=30336
192.168.0.100	TCP	113	49524→8888 [PSH, ACK] Seq=1254 Ack=139 Win=3
172.17.0.2	TCP	113	47512→8888 [PSH, ACK] Seq=1254 Ack=139 Win=3
192.168.0.100	TCP	164	49524→8888 [PSH, ACK] Seq=1301 Ack=139 Win=3
172.17.0.3	TCP	66	8888→49524 [ACK] Seq=139 Ack=1399 Win=31872
172.17.0.2	TCP	164	47512→8888 [PSH, ACK] Seq=1301 Ack=139 Win=3
172.17.0.1	TCP	66	8888→47512 [ACK] Seq=139 Ack=1399 Win=31872
172.17.0.1	TCP	328	8888→47512 [PSH, ACK] Seq=139 Ack=1399 Win=3
172.17.0.1	TCP	703	8888→47512 [PSH, ACK] Seq=401 Ack=1399 Win=3
172.17.0.2	TCP	66	47512→8888 [ACK] Seq=1399 Ack=1038 Win=32640
172.17.0.3	TCP	328	8888→49524 [PSH, ACK] Seq=139 Ack=1399 Win=3
172.17.0.3	TCP	703	8888→49524 [PSH, ACK] Seq=401 Ack=1399 Win=3
192.168.0.100	TCP	66	49524→8888 [ACK] Seq=1399 Ack=1038 Win=32640
192.168.0.100	TCP	93	49524→8888 [PSH, ACK] Seq=1399 Ack=1038 Win=
172.17.0.2	TCP	93	47512→8888 [PSH, ACK] Seq=1399 Ack=1038 Win=
172.17.0.1	TCP	93	8888→47512 [PSH, ACK] Seq=1038 Ack=1426 Win=
172.17.0.3	TCP	93	8888→49524 [PSH, ACK] Seq=1038 Ack=1426 Win=

Abbildung 36: Verschlüsselter Mitschnitt von Wireshark

2. Signierung
   Da es nicht nur wichtig ist, dass die übertragenen Daten nicht gelesen, sondern auch keine Daten bearbeitet oder eingeschleust werden, testen wir nun die Überprüfung der Signatur.

   Dazu erstellen wir auf dem Ubuntu-Container neue RSA-Schlüssel und starten stunnel neu.

   Schicken wir nun die gleiche Anfrage, erhalten wir als Antwort:

```
1 $ curl localhost:80
2 stunnel: ...: Service [...] accepted connection from ...
3 ...
4 stunnel: ...: Certificate accepted: ...
5 stunnel: ...: SSL_connect: ... alert unknown ca
6 ...
7 curl: (56) Recv failure: Connection reset by peer
```

   Wie man in Zeile 2 sieht, wird die Verbindung aufgebaut und in Zeile 4

das Zertifikat des Servers akzeptiert. Trotzdem wird die Verbindung mit der Fehlermeldung in Zeile 5 abgebrochen und keine Daten übertragen.

Erzeugen wir dagegen auf dem Server (nginx) neue Schlüssel und spielen auf dem Client (Ubuntu) wieder die alten ein, erhalten wir auf die Anfrage folgende Antwort:

```
1 $ curl localhost:80
2 stunnel: ...: Service [...] accepted connection from ...
3 ...
4 stunnel: ...: Certificate check failed: ...
5 stunnel: ...: SSL_connect: ... certificate verify failed
6 ...
7 curl: (56) Recv failure: Connection reset by peer
```

Wie zuvor wird die Verbindung aufgebaut, allerdings wird hier bereits das Zertifikat des Servers abgelehnt und die Übertragung abgebrochen.

5.4.3 L03: nginx

Zum Testen der Sicherheit von nginx wird das Online-Tool, welches von Mozilla empfohlene wird[25], der Seite `https://www.ssllabs.com` verwendet. Dafür wird die Konfiguration, vorübergehend auf einem Webserver gestartet und anschließend mit dem Tool getestet. Um dabei die, hier nicht relevante, Fehlermeldung eines selbst signierten Zertifikats zu vermeiden, wurde dafür ein signiertes Zertifikat verwendet.

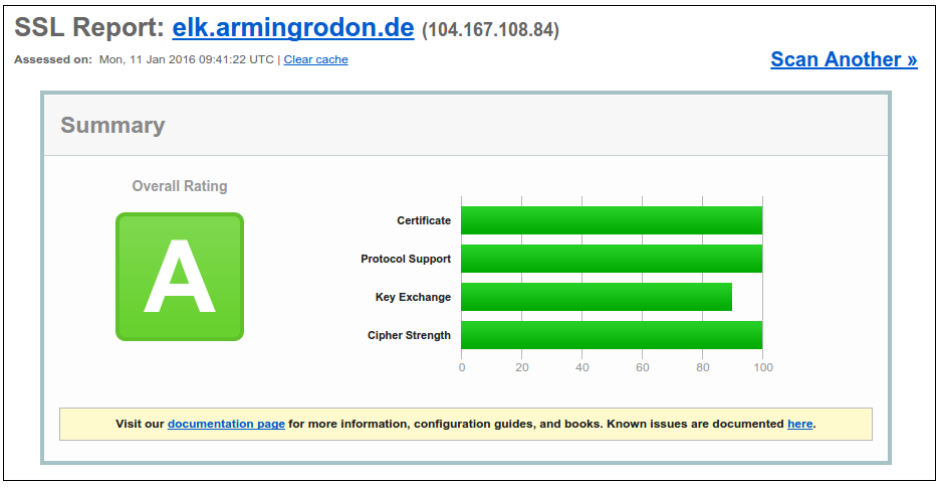


Abbildung 37: Verifizierung mit ssllabs

Die fehlenden Prozentpunkte ergeben sich daraus, dass nginx bei der Auslieferung von Fehlerseiten keine zusätzlichen `add_header`-Anweisungen übernimmt und der Test ohne Authentifizierung auf einer 401-Seite landet.
Dies lässt sich leicht überprüfen, indem die Konfiguration für eine andere Domain ohne Authentifizierung eingerichtet wird.

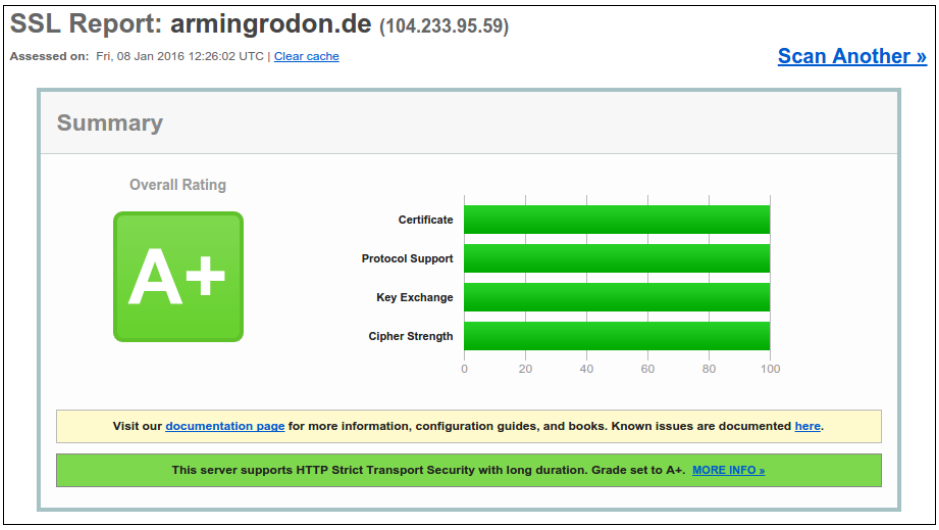


Abbildung 38: Perfekte Bewertung bei ssllabs

Hier werden auch der Header für Strict Transport Security (HSTS) richtig gesetzt. Um abschließend sicher zu gehen, dass sie auch mit korrekter Authentifizierung gesetzt werden, kann man mit `curl` den Header der Domain anfragen und die Authentifizierungsdaten mitsenden.

```
~ > Downloads > curl -s -D- https://elk.armingrodon.de/
HTTP/1.1 401 Unauthorized
Server: nginx/1.9.9
Date: Mon, 11 Jan 2016 10:37:15 GMT
Content-Type: text/html
Content-Length: 194
Connection: keep-alive
WWW-Authenticate: Basic realm="Restricted"

~ > Downloads > curl --user guest:[redacted] -s -D- https://elk.armingrodon.de/ | grep Strict
Strict-Transport-Security: max-age=31536000; includeSubdomains; preload
~ > Downloads >
```

Abbildung 39: Test auf Strict Transport Security

Wie man sieht, ist der Header für HSTS ohne Authentifizierung, wie auch schon vermutet, nicht gesetzt. Sobald allerdings die korrekten Benutzerdaten

gesendet werden, wird auch der Header korrekt gesetzt. Ein angemeldeter Benutzer ist damit, wie auch ein Besucher einer Seite ohne Authentifizierung, nach den aktuellen Maßstäben bestens geschützt.

5.4.4 L04: Verschlüsselte Persistierung

Um die Verschlüsselung von eCryptfs zu testen erstellen wir, wie in Kapitel Konfiguration von Logstash beschrieben, einen Schlüssel für eCryptfs und mounten damit einen Ordner.
Anschließend schreiben wir eine Textdatei und geben zur Überprüfung, dass die Daten korrekt gespeichert werden den Inhalt wieder aus. Danach unmounten wir den Ordner und geben den Inhalt erneut aus.

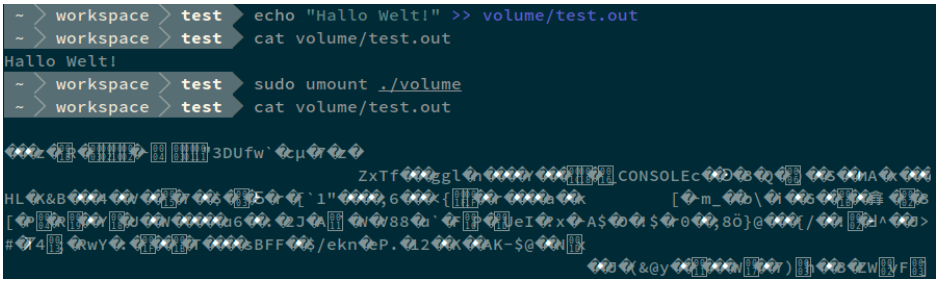


Abbildung 40: Testen von eCryptfs

5.5 Fazit

Obwohl das das Projekt insgesamt um einiges arbeitsintensiver war als anfangs angenommen, sind wir aus eigener Sicht zu einem sehr zufriedenstellenden Ergebnis gekommen. Mit ELK wurde eine zweite Möglichkeit eines Überwachungsdienstes umgesetzt, welche zwar im Gegensatz zur selbst programmierten Lösung eventuell an Grenzen in den Einsatzgebieten stoßen könnte, dafür aber unerreichte Analyse- und Visualisierungsmöglichkeiten bietet.
Es wurde gezeigt, dass auch ein Produkt, welches nicht mit Sicherheit als Hauptaugenmerk entwickelt wurde, mit genug Arbeit nach Außen hin abgesichert werden und dadurch hohen Sicherheitsstandards genügen kann.
Im Laufe der Absicherung habe ich viele neue Werkzeuge kennen gelernt, auch wenn sich einige davon nicht für dieses Projekt geeignet haben und

deshalb nicht zum Einsatz kamen, welche bei späteren Projekten im Sicherheitsumfeld sehr hilfreich sein können.
Im Allgemeinen hat sich der Arbeitsaufwand zumindest, alleine schon durch den Erkenntnisgewinn, auf jeden Fall gelohnt.

5.6 Ausblick

Da das Aufsetzen der Architektur so viel Zeit in Anspruch nahm, war es leider nicht möglich alle geplanten Aktionen umzusetzen. Diese sind deshalb für eine spätere Iteration geplant.
Allen voran ist dies die Replizierung von ELK. Dies ist besonders dadurch interessant, da die einzelnen Komponenten sehr gut horizontal skalieren und damit neben Ausfallsicherheit und Lastverteilung auch eine schnellere Datenverarbeitung möglich ist.
Ein weiterer Punkt ist die Umsetzung von OAuth für Kibana oder die Implementierung eines komplett anderen Authentifizierungsdienstes für die gesamte Anwendung um nicht auf die Datenverarbeitung durch ein fremdes Unternehmen angewiesen zu sein.
Als letzter Punkt ist hier noch die Auswertung der Daten aller anderen Teams zu nennen. Dies war leider zeitlich und organisatorisch nicht möglich. Durch das Testen mit Daten von SSH und iptables hat sich aber gezeigt, dass die zentrale Auswertung von verteilten Diensten Informationen liefert, die sonst nicht zugänglich wären. Zusätzlich bieten sie durch die grafische Aufarbeitung die Möglichkeit, Zusammenhänge zu erkennen, die sonst durch die Menge an Informationen und die zeitliche Abfolge nicht auf einen Blick zu verarbeiten wären.

Literatur

[1] "What is docker?." https://www.docker.com/what-docker. (Aufgerufen am: 12.01.2016).

[2] "Dockerfile reference." "https://docs.docker.com/engine/reference/builder/".

[3] "Overview of docker compose." "https://docs.docker.com/compose/".

[4] "boot2docker." http://boot2docker.io/. (Aufgerufen am: 13.01.2016).

[5] "v0.5.0." https://github.com/docker/machine/releases/tag/v0.5.0. (Aufgerufen am: 13.01.2016).

[6] "The perfect little place to run docker." http://rancher.com/rancher-os/. (Aufgerufen am: 13.01.2016).

[7] "redis.io." http://redis.io/. (Aufgerufen am: 12.01.2016).

[8] "Apache tomcat." https://tomcat.apache.org/. (Aufgerufen am: 12.01.2016).

[9] "Small. simple. secure." http://www.alpinelinux.org/.

[10] "Elastic Dokumentation." https://www.elastic.co/guide. (Aufgerufen am: 05.01.2016).

[11] "stunnel Dokumentation." https://www.stunnel.org. (Aufgerufen am: 05.01.2016).

[12] "nginx Dokumentation." http://nginx.org/en/docs. (Aufgerufen am: 05.01.2016).

[13] "ecryptfs Manpage." http://manpages.ubuntu.com/manpages/wily/en/man7/ecryptfs.7.html. (Aufgerufen am: 05.01.2016).

[14] "IT-Grundschutz-Katalog: Elementare Gefährdungen." https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/g/g00/g00.html. (Aufgerufen am: 05.01.2016).

[15] “Redis security.” "http://redis.io/topics/security".

[16] “Securing tomcat.” "https://www.owasp.org/index.php/Securing_tomcat".

[17] “7.2. advantages of using docker.” "TheOAuth2.0AuthorizationFramework".

[18] “Elastic Shield Produktseite.” https://www.elastic.co/products/shield. (Aufgerufen am: 05.01.2016).

[19] “Docker Guide: Verlinken von Containern.” https://docs.docker.com/v1.8/userguide/dockerlinks. (Aufgerufen am: 05.01.2016).

[20] “Docker Dokumentation zu Sicherheit.” https://docs.docker.com/engine/security. (Aufgerufen am: 05.01.2016).

[21] “Tutorial: Authentifikation mit nginx.” https://www.digitalocean.com/community/tutorials/how-to-set-up-http-authentication-with-nginx-on-ubuntu-12-10. (Aufgerufen am: 05.01.2016).

[22] “Tutorial: Selbstsignierte Zertifizierung für nginx.” https://www.digitalocean.com/community/tutorials/how-to-create-an-ssl-certificate-on-nginx-for-ubuntu-14-04. (Aufgerufen am: 05.01.2016).

[23] “Tutorial: nginx als reverse Proxy.” https://www.digitalocean.com/community/tutorials/how-to-configure-nginx-as-a-web-server-and-reverse-proxy-for-apache-on-one-ubuntu-14-04-droplet. (Aufgerufen am: 05.01.2016).

[24] “Tutorial: nginx härten.” https://raymii.org/s/tutorials/Strong_SSL_Security_On_nginx.html. (Aufgerufen am: 05.01.2016).

[25] “MoziallaWiki: Security/Server Side TLS.” https://wiki.mozilla.org/Security/Server_Side_TLS. (Aufgerufen am: 12.01.2016).

[26] "Tecmint: The Ultimate Guide to Secure, Harden and Improve Performance of Nginx Web Server." http://www.tecmint.com/nginx-web-server-security-hardening-and-performance-tips/. (Aufgerufen am: 12.01.2016).

[27] "Elastic Dokumentation: grok." https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html. (Aufgerufen am: 05.01.2016).

[28] "OWASP: SELinux, Apache, and Tomcat - A Securely Implemented Web Application Server." https://www.owasp.org/images/0/01/Secure_Web_App_Server_McRee_OWASP.pdf. (Aufgerufen am: 12.01.2016).