Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Martin Staib

Erkennung und Visualisierung von verwundbaren Systemen anhand von passiv aufgezeichneten Netzwerk-Daten

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Martin Staib

Erkennung und Visualisierung von verwundbaren Systemen anhand von passiv aufgezeichneten Netzwerk-Daten

Bachelorarbeit eingereicht im Rahmen Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Klaus-Peter Kossakowski
Zweitgutachter : Prof. Dr.-Ing Olaf Zukunft

Abgegeben am 18.06.2018

Martin Staib

Thema der Arbeit

Erkennung und Visualisierung von verwundbaren Systemen anhand von passiv aufgezeichneten Netzwerk-Daten

Stichworte

IT-Sicherheit, passive Netzwerkdaten, Sensor, CVE-Datenbank, Schwachstellen

Kurzzusammenfassung

In dieser Arbeit wird ein Prototyp entwickelt, welcher unverschlüsselten Netzwerkverkehr passiv aufzeichnen, anschließend die aufgezeichneten Daten auswerten und die Ergebnisse der Auswertung anzeigen kann. Die Daten werden hierbei mithilfe von Sensoren gesammelt und mit einer Konsolenanwendung ausgewertet und angezeigt. Mit den Sensoren wird versucht, aus dem Netzwerkverkehr die Daten der Anwendungsschicht herauszufiltern. Wenn dies möglich ist, wird versucht, aus diesen Daten Servicenamen und -versionen der jeweiligen Anwendungsschichtprogramme zu extrahieren. Ist auch dies möglich, werden nun die extrahierten Servicenamen und -versionen gegen eine CVE-Datenbank geprüft. Somit können verwundbare Services, also solche, die Schwachstellen enthalten, erkannt werden. Anschließend werden die Ergebnisse der Prüfung angezeigt.

Martin Staib

Title of the paper

Detection and visualization of vulnerable systems based on passive logged network data

Keywords

IT security, passive network data, sensor, cve-database, vulnerability

Abstract

In this thesis a prototype is developed, which passively records clear text network data, subsequently evaluates the recorded data and shows the results of the evaluation. The network data is recorded by sensors, the evaluation and the display of the results is done by a console application. The purpose is to filter application-layer data from the network data with the sensors. If this is possible, it is attempted to extract service names and versions from each application-layer software. If this also is possible, the extracted service names and versions are checked against a CVE-database. Therefore it is possible to detect vulnerable services. Afterwards the results of the CVE-check are displayed.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel wird der Rahmen der Arbeit erläutert. Zunächst wird die Motivation für die Arbeit dargelegt. Anschließend werden die Ziele der Arbeit sowie auch Abgrenzungen gegenüber diesen Zielen definiert. Die Zielgruppe dieser Arbeit wird ebenfalls definiert, abschließend wird der Aufbau der Arbeit skizziert.

1.1 Motivation

In Zeiten fortschreitender Vernetzung werden immer mehr Dienste und Arbeitsprozesse über das Internet angeboten und abgewickelt. Mit dieser steigenden Menge an Aufgaben nimmt auch die Menge an Daten zu, insbesondere die Menge an sensiblen Daten. Dienste, die mit solchen sensiblen Daten in Berührung kommen, müssen diese Daten dementsprechend absichern und vor unbefugtem Zugriff schützen. Auch die Dienste selbst müssen geschützt werden. Diese Aufgaben gehören zu den Kernaufgaben der IT-Sicherheit. Hierzu stehen ihr nun verschiedene Konzepte und Strategien zur Verfügung. Angefangen bei der reinen physischen oder logischen Separierung einzelner Netzwerke, über strategische Lösungsansätze wie etwa entmilitarisierte Zonen, bis hin zu technischen Aspekten wie etwa Verschlüsselung, sind die Möglichkeiten, Daten und Dienste zu schützen, vielfältig und breit gefächert. In der Praxis kommen diese verschiedenen Ansätze meistens kombiniert zur

Anwendung. Diese Kombination erhöht widerum die Sicherheit der zu schützenden Dienste und Daten. Ein weiterer Aspekt der Dienst- und Datensicherheit ist, dass fast alle Netzwerkdienste verschiedene Modi anbeiten, in denen sie betrieben werden können. Als Beispiel kann hier etwa ein Webserver herangezogen werden, der im unverschlüsselten Modus (Port 80, HTTP) oder im verschlüsselten Modus (Port 443, HTTPS) arbeiten kann. In welchem Modus ein Dienst arbeitet, kann also erhebliche Konsequenzen für seine Sicherheit und die Sicherheit seiner Daten haben.

Diese Arbeit will eine Möglichkeit aufzeigen, Dienste zu erkennen, welche im unverschlüsselten Modus arbeiten. Mit dem Erkennen dieser Dienste können potentielle Sicherheitslücken geschlossen und die Sicherheit des Dienstes und seiner Daten erhöht werden. Dies kann etwa geschehen, indem der Dienst als unverschlüsselt arbeitend erkannt und infolgedessen ein Administrator informiert wird, welcher den Dienst dann auf den verschlüsselten Modus umstellt. Ist dieses Vorgehen nicht gewollt oder möglich, kann noch immer eine zweite Strategie verfolgt werden. Wird ein Dienst als unverschlüsselt arbeitend erkannt und soll auch weiterhin in diesem Modus betrieben werden, kann wenigstens die Version des Dienstes aktuell gehalten oder ein Überblick über bekannte Schwachstellen dieses Dienstes gewonnen werden. Hierzu werden sein Name und seine Version gegen eine CVE-Datenbank geprüft und der Administrator kann die Dienstversion anschließend aktualisieren. Ist die Version aktuell und die Prüfung gegen die CVE-Datenbank hat Schwachstellen für den Dienst ergeben, so sind diese dem Administrator nun bekannt. Der Dienstmodus soll zwar nicht geändert werden, aber der Administrator kann nun eventuell andere Maßnahmen zur Erhöhung der Dienstsicherheit treffen.

1.2 Ziel und Abgrenzung

Das Ziel dieser Arbeit ist die Entwicklung eines Softwareprototypen, mit dem unverschlüsselter Netzwerkverkehr passiv aufgezeichnet werden kann. Hierbei wird versucht, aus den Daten der Anwendungsschicht-programme deren jeweilige Namen und Versionen zu extrahieren. Diese zu extrahierenden Informationen sollen in mehreren Netzwerken gleichzeitig gesammelt werden und an einer zentralen Stelle zusammenlaufen, wo sie verwaltet werden. Dem Benutzer des Prototypen, etwa einem Administrator, soll diese zentrale Verwaltungsstelle eine Schnittstelle zu ihrer Bedienung anbieten. Beispielhaft soll an ausgewählten Services gezeigt werden, ob und wie das beschriebene Vorhaben bewerkstelligt werden kann.

Es ist weder das Ziel dieser Arbeit, den vollständigen Bereich aller Anwendungsschichtprogramme abzudecken, noch auch nur ein Beispiel für jeden Bereich an Anwendungssoftware zu liefern. Ebenfalls ist es nicht Ziel dieser Arbeit zu versuchen, aus verschlüsseltem Netzwerkverkehr Signaturen derart herauszuarbeiten, dass aus ihnen Rückschlüsse auf die dahinter arbeitenden Services gezogen werden können.

1.3 Zielgruppe

Diese Arbeit richtet sich an Studenten der Informatik im Allgemeinen und im Spezielleren an solche, die IT-Sicherheit als Studienspezialisierung gewählt haben. Desweiteren richtet sich diese Arbeit an Personen, die bereits im Bereich der IT-Sicherheit tätig sind, insbesondere an Netzwerk-administratoren oder Mitarbeiter von IT-Abteilungen.

1.4 Aufbau der Arbeit

Im zweiten Kapitel werden die Grundlagen zum technischen Verständnis dieser Arbeit dargelegt. Dem Leser eventuell nicht hinlänglich bekannte, für diese Arbeit spezifische Sachverhalte werden ihm hier kurz erläutert und an die Hand gegeben. Abschließend wird ein kurzer Überblick über verwandte Arbeiten gegeben.

Im dritten Kapitel wird ein allgemeines Lösungskonzept für die Problemstellung dieser Arbeit entwickelt. Hier wird zunächst eine Anforderungsanalyse für den zu entwickelnden Prototypen durchgeführt. Auf Basis dieser Anforderungsanalyse werden Architekturentscheidungen getroffen und begründet. Abschließend wird ein Systementwurf für den zu entwickelnden Prototypen erarbeitet.

Das vierte Kapitel behandelt die Implementierung des Prototypen. Zunächst werden konkrete Technologieentscheidungen getroffen und begründet. Anschließend wird die Umsetzung der Implementierung des Prototypen ausführlich dargelegt. Zum Abschluss dieses Kapitels wird die Infrastruktur erklärt, welche zum Zwecke der Validierung des Prototypen, wie auch bei der Entwicklung des Prototypen selbst, als Test- und Arbeitsumgebung eingesetzt wurde.

Im fünften und letzten Kapitel dieser Arbeit werden abschließend die erarbeiteten Ergebnisse präsentiert. Zunächst werden in einer Zusammenfassung die erreichten Ziele diskutiert und kurz mit den gesteckten Zielen abgeglichen. Anschließend wird ein Ausblick auf weitere Arbeiten und Ideen gegeben, welche auf dieser Arbeit aufbauen könnten.

2 Grundlagen

In diesem Kapitel werden die technischen Grundlagen für das Verständnis dieser Arbeit gelegt. Eher spezifische, aber für diese Arbeit wichtige Sachverhalte werden hier erklärt. Zunächst werden die Schutzziele der Informationssicherheit erläutert, ehe Sensoren und Sniffer erklärt werden. Anschließend wird in die Themen Netzwerkdaten und Standardports sowie Vulnerability und CVE eingeführt. Weiterhin werden die Docker Container Platform, Manual Pages und Message Broker, das JSON-Datenaustauschformat und ORM erklärt. Abschließend wird kurz, anhand von verwandten Arbeiten, der Stand der Technik aufgezeigt.

2.1 Schutzziele der IT-Sicherheit

Die klassischen Schutzziele der IT-Sicherheit sind Vertraulichkeit, Verfügbarkeit und Integrität. Sie beziehen sich auf Daten und Informationen von IT-Systemen. Konkret meinen die einzelnen Ziele Vertraulichkeit von Informationen, Verfügbarkeit von IT-Systemen und somit deren Daten sowie Datenintegrität. Hierbei geht es darum, nur autorisierten Instanzen Zugriff auf vertrauliche Informationen zu gewähren. Haben diese Instanzen sich authentifiziert und somit als autorisiert erwiesen, soll ihnen der Zugriff auf ihre Daten gewährleistet werden. Eine Datenmanipulation soll ebenfalls nur durch autorisierte Instanzen erfolgen können. [vgl. Eckert, 2013, S.7 ff]

- **Integrität:** „Wir sagen, dass das System die Datenintegrität (engl. *integrity*) gewährleistet, wenn es Subjekten nicht möglich ist, die zu schützenden Daten unautorisiert und unbemerkt zu manipulieren.“ [Eckert, 2013, S.9]

- **Vertraulichkeit:** „Wir sagen, dass das System die Informationsvertraulichkeit (engl. *confidentiality*) gewährleistet, wenn es keine unautorisierte Informationsgewinnung ermöglicht.“ [Eckert, 2013, S.10]

- **Verfügbarkeit:** „Wir sagen, dass das System die Verfügbarkeit (engl. *availability*) gewährleistet, wenn authentifizierte und autorisierte Subjekte in der Wahrnehmung ihrer Berechtigungen nicht unautorisiert beeinträchtigt werden können.“ [Eckert, 2013, S.12]

2.2 Sensoren und Sniffer

Sensor: Ein Sensor ist ein Detektor zum Erfassen von Eigenschaften bestimmter Systeme. Es gibt aktive und passive Sensoren. Im Allgemeinen wird von Sensoren eher im Kontext physikalischer Messgrößen gesprochen. Im Kontext der Informatik und speziell im Kontext der IDS (Intrusion Detection Systeme) spricht man von Sensoren als Detektoren, welche in Netzwerken eingesetzt werden. Sie dienen dazu, die Netzwerkdaten zu analysieren und gegen vorher definierte Signaturen abzugleichen. Hierbei unterscheidet man zwischen zwei Varianten von Sensoren, dem *Inline-Sensor* und dem *Out-of-Line-Sensor*.

- **Inline-Sensor:** „Ein Inline-Sensor sitzt direkt im Datenpfad der zu überwachenden Pakete. Typischerweise ist der Sensor in solchen Fällen auf einer Bridge oder einem Router untergebracht, die neben ihrer eigentlichen Aufgabe auch Bestandteil des IDS sind. Der Sensor muss die Pakete selbst also aktiv zum Ziel weiterbefördern.“ [Kappes, 2013, S.233]

- **Out-of-Line-Sensor:** „Ein Out-of-Line-Sensor befindet sich nicht direkt im Datenpfad, sondern erhält Kopien der zu überwachenden Pakete. Dies kann technisch auf verschiedene Arten realisiert werden. Beispielsweise bieten einige Switches die Möglichkeit, einen Mirrorport einzurichten, an dem die aus dem Switch an einen oder mehrere Ports ausgehenden Daten zusätzlich auf den Mirrorport weitergeleitet („gespiegelt“) werden. Auf diese Weise kann der über den Switch abgewickelte Verkehr am Mirrorport durch einen Sensor analysiert werden.“ [Kappes, 2013, S.233]

Inline-Sensoren besitzen gegenüber *Out-of-Line-Sensoren* den Vorteil, dass sie direkt an der Weiterleitung der Daten beteiligt sind und somit bereits aktiv eingreifen können, sobald sie verdächtige Datenpakete detektiert haben. Allerdings ist dieser Vorteil auch gleichzeitig ein großer Nachteil. Da *Inline-Sensoren* nämlich dafür zuständig sind, Datenpakete zu befördern, hängt die gesamte Weiterbeförderung dieser Datenpakete an der korrekten Arbeitsweise der Sensoren. Selbst wenn sie korrekt arbeiten, verbrauchen sie eine gewisse Rechenkapazität für das Untersuchen der Datenpakete und das anschließende Abgleichen gegen die Signaturen. Da nun Sensoren häufig an zentralen Netzwerkpunkten eingesetzt werden, ist klar, dass solche *Inline-Sensoren* schnell überlastet werden und somit das gesamte Netzwerk zum Ausfall bringen können. Auch *Out-of-Line-Sensoren* besitzen Nachteile, allerdings sind diese als deutlich geringer einzuordnen. [vgl. Kappes, 2013, S.233 ff]

Sniffer: Ein Sniffer ist ein Werkzeug zur Netzwerkanalyse, mit dem der Netzwerkverkehr mitgelesen und aufgezeichnet werden kann. Es gibt zwei verschiedene Modi, in denen ein Sniffer betrieben werden kann. Im *Non-Promiscuous-Mode* arbeitet der Sniffer auf einem einzelnen Computer und zeichnet nur den Netzwerkverkehr dieses Computers auf. Im *Promiscuous-Mode* können Sniffer den Datenverkehr eines gesamten Netzwerkes abfangen. [EASY-NETWORK, 2018, SNIFFER]
Sniffer können als Bestandteile von Sensoren eingesetzt werden. Bekannte Beispiele für Sniffer sind etwa *wireshark*, *tcpdump* oder *Capsa*.

2.3 Netzwerkdaten und Standardports

Netzwerkdaten: Es exisiteren unterschiedliche Arten von Netzwerkdaten, abhängig davon, aus welcher Netzwerkschicht diese Daten stammen. Das *ISO/OSI-Referenzmodell* beschreibt die einzelnen Netzwerkschichten. Mit diesen definierten Netzwerkschichten werden auch die jeweils aus ihnen stammenden Daten klassifiziert. Abbildung 2.1 zeigt die Netzwerkschichten nach dem *ISO/OSI-Referenzmodell* mit den dazugehörigen Netzwerkdaten.

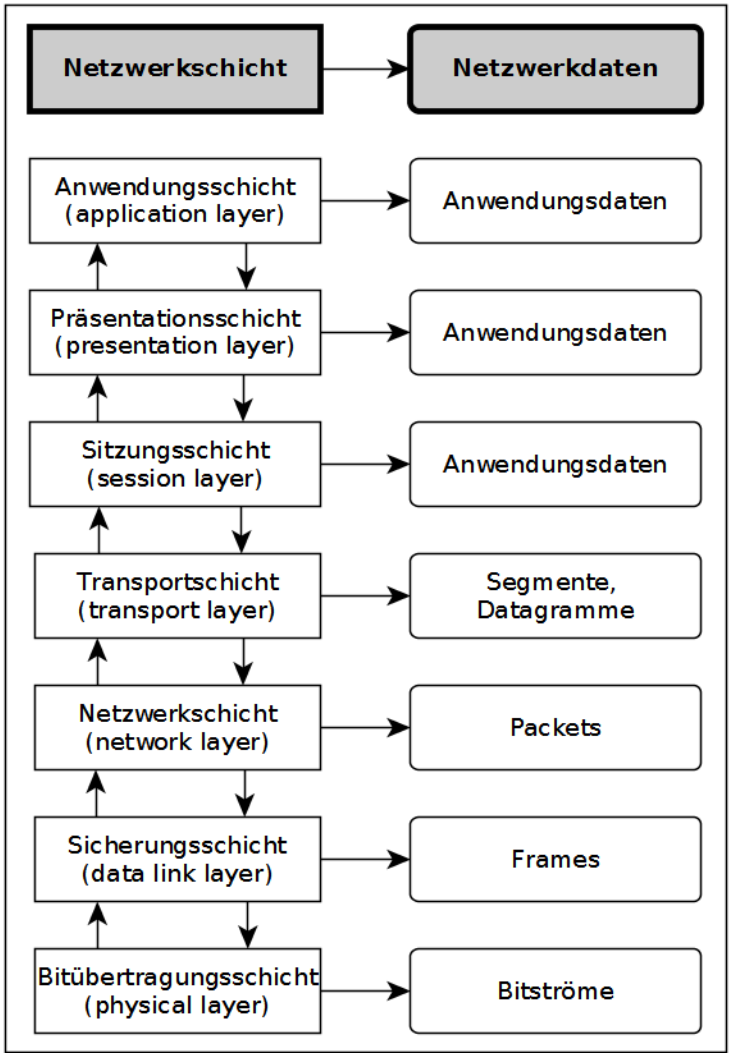

Abbildung 2.1: Netzwerkschichten und Netzwerkdaten [vgl. Eckert, 2013, S.97 ff]

Passive Netzwerkdaten sind solche, die durch passives Monitoring von Netzwerken gewonnen werden. Bei diesem passiven Monitoring werden lediglich die schon im Netzwerk vorhandenen Datenpakete aufgezeichnet und analysiert. Es werden keine künstlichen Testdaten zusätzlich in das zu überwachende Netzwerk injiziert. Im Gegensatz dazu gibt es auch aktive Netzwerkdaten, welche aus aktivem Netzwerkmonitoring resultieren. Hierbei werden zu Analysezwecken zusätzlich Testdaten in das zu überwachende Netzwerk eingebracht. [TSPRO, 2018, NETMON]

Standardports: Die IANA (*Internet Assigned Numbers Authority)* veröffentlicht eine Liste, in welcher einzelnen Ports eindeutig bestimmte Protokolle zugeordnet werden. Diese Liste ist ein Standard und umfasst die Ports mit den Nummern 0–1023. Alle Ports mit höheren Nummern sind nicht offiziell in dieser Liste der sogenannten *Sytem Ports* oder *well-known ports* registriert. [IANA, 2018, REGISTRY]
Tabelle 2.1 zeigt einen Auszug aus der Liste der *System Ports*. So wird beispielsweise ein HTTP-Webserver standardmäßig auf Port 80 betrieben. Diese Zuordnung von Protokollen auf Ports ist aber nicht bindend. Zwar sind die *System Ports* jeweils für bestimmte Protokolle blockiert, jedoch können die Protokolle ihrerseits auf andere Ports, welche außerhalb der *System Ports* liegen, ausweichen. So kann ein HTTP-Webserver beispielsweise auch auf Port 50.000 betrieben werden.

Servicename	Portnummer	Transportprotokoll	Beschreibung
FTP	21	TCP	File Transfer Protocol
SSH	22	TCP	The Secure Shell Protocol
TELNET	23	TCP	The Telnet Protocol
SMTP	25	TCP	Simple Mail Transfer Protocol
HTTP	80	TCP	Hypertext Transfer Protocol

Tabelle 2.1: Auszug aus der IANA-Liste der Standardports [IANA, 2018, REGISTRY]

2.4 Vulnerability und CVE

Vulnerability: Mit Vulnerability bezeichnet man eine Schwachstelle oder einen Defekt im Design, in der Implementation oder im Betrieb einer Software. Schwachstellen können unbemerkt bleiben und müssen nicht zwangsläufig Angriffe auf die Software, in der sie auftreten, zur Folge haben. Allerdings können sie Einfallsvektoren für Angreifer darstellen, insofern den Angreifern die Schwachstellen bekannt sind. [IETF, 2000, RFC2828]

CVE: CVE (*Common Vulnerabilities and Exposures*) ist ein Industriestandard zur einheitlichen Benennung von aufgedeckten Schwachstellen in Computersystemen. Ziel dieses Standards ist es, Schwachstellen eindeutig zu erfassen und zu katalogisieren, um Softwareherstellern eindeutige Referenzen anzubieten, mithilfe derer sie Schwachstellen in ihrer eigenen Software schließen können. Das eindeutige Erfassen der Schwachstellen geschieht mithilfe sogenannter CVE-Nummern (CVE-IDs). Eine jede erkannte Schwachstelle bekommt ihre eindeutige CVE-ID zugewiesen und wird als eigenständiger Eintrag in der Liste der CVEs abgespeichert. Seit 1999 wird diese Liste von der *Mitre Corporation* in Zusammenarbeit mit den *CVE Numbering Authorities* verwaltet und herausgegeben. Zum Zwecke der Einsicht in die CVE-Liste existiert eine sogenannte CVE-Datenbank. Diese ist online über die Webseite der *Mitre Corporation* verfügbar. Ebenfalls besteht die Möglichkeit, eine aktuelle Version der CVE-Datenbank von der Webseite der *Mitre Corporation* herunterzuladen. [MITRE, 2018, CVE]

Die Einträge aus der CVE-Liste bestehen aus folgenden Feldern:

- CVE-ID der Schwachstelle bestehend aus:
  - Präfix + Jahr der Publizierung + Sequenznummer
- Beschreibung der Schwachstelle
- Referenzen zur Schwachstelle
- zugeordnete CVE Numbering Authority (CNA)
- Datum des Eintrags in die Liste

Abbildung 2.2 zeigt beispielhaft einen solchen CVE-Listeneintrag.

CVE-ID: CVE-2016-1627

Beschreibung: The Developer Tools (aka DevTools) subsystem in Google Chrome before 48.0.2564.109 does not validate URL schemes and ensure that the remoteBase parameter is associated with a chrome-devtools-frontend.appspot.com URL, which allows remote attackers to bypass intended access restrictions via a crafted URL, related to browser/devtools/devtools_ui_bindings.cc and WebKit/Source/devtools/front_end/Runtime.js.

Referenzen:

- CONFIRM:http://googlechromereleases.blogspot.com/2016/02/stable-channel-update_9.html
- CONFIRM:https://code.google.com/p/chromium/issues/detail?id=571121
- CONFIRM:https://code.google.com/p/chromium/issues/detail?id=585517
- CONFIRM:https://codereview.chromium.org/1586903002
- DEBIAN:DSA-3486
- URL:http://www.debian.org/security/2016/dsa-3486
- GENTOO:GLSA-201603-09
- URL:https://security.gentoo.org/glsa/201603-09
- REDHAT:RHSA-2016:0241
- URL:http://rhn.redhat.com/errata/RHSA-2016-0241.html
- SUSE:openSUSE-SU-2016:0518
- URL:http://lists.opensuse.org/opensuse-updates/2016-02/msg00119.html
- SUSE:openSUSE-SU-2016:0491
- URL:http://lists.opensuse.org/opensuse-updates/2016-02/msg00104.html
- BID:83125
- URL:http://www.securityfocus.com/bid/83125
- SECTRACK:1035183
- URL:http://www.securitytracker.com/id/1035183

zugeordnete CNA: N/A

Datum Listeneintrag: 20160112

Abbildung 2.2: Beispiel eines CVE-Listeneintrags [MITRE, 2018, CVENAME]

2.5 Docker Container Platform

Docker ist eine Firma, welche eine Container-Platform-Software anbietet. In einer solchen Container-Platform-Software können Anwendungen isoliert in sogenannten Containern betrieben werden. Es handelt sich

hierbei um eine Art der Virtualisierung. In den Containern können gezielt bestimmte Pakete und Programme installiert werden und die Container selbst lassen sich widerum leicht als Datei transportieren und installieren. Containervirtualisierung gewährleistet den getrennten Betrieb einzelner Ressourcen sowie die getrennte Verwaltung dieser Ressourcen auf der Hostmaschine. Jeder Container bekommt vom Hostsystem Zugriff auf ausgewählte Bibliotheken und Dateien. Die Container selbst sind keine kompletten virtuellen Maschinen im Sinne eines eigenständigen Betriebssystems. Sie bekommen aus Sicherheitsgründen lediglich einen sehr eingeschränkten Zugriff auf die Kernelfunktionen des Hostbetriebssystems. [DOCKER, 2018, WHAT-IS]

2.6 Manual Page und Message Broker

Manual Page: Eine Manual Page ist eine Hilfe- und Dokumentationsseite für Programme in unixoiden Systemen. Oft wird der Begriff auch im Sinne eines Handbuches für einzelne Programme genutzt. Da Manual Pages aus dem Unixumfeld stammen, werden sie üblicherweise aus einer Kommandozeile (*shell*) heraus aufgerufen. Der dafür gängige Befehl lautet `man <Programmname>`. [MANPAGE, 2011, HISTORY]

Message Broker: Message Broker sind zwischen verschiedene andere Programme geschaltete Programm-Module mit der Aufgabe, Nachrichten zwischen diesen anderen Programmen weiterzuleiten. Hierbei implementieren Message Broker formale Nachrichtenprotokolle wie etwa das AMQP (*Advanced Message Queuing Protocol*). Um diese Aufgabe erfüllen zu können, sind sie in der Lage, Nachrichten zu validieren, zu transformieren und weiterzuleiten. Sie sind feste Bestandteile von Telekommunikations- und Computernetzwerken. Beispiele für bekannte Message Broker sind *RabbitMQ*, *Oracle Message Broker* oder *Microsoft Azure Service Bus*. Der Begriff Message Broker bezeichnet ebenfalls ein Architekturmuster für die Nachrichtenvalidation, -transformation und -weiterleitung. [3PILLAR, 2018, BROKER]

2.7 JSON und ORM

JSON: JSON (*JavaScript Object Notation*) ist ein Format zum Austausch von Daten zwischen Anwendungen. Es wurde nach den Vorgaben entwickelt, für Menschen leicht lesbar und für Maschinen leicht generierbar und parsbar zu sein. Das JSON-Format wird im RFC 8259 spezifiziert. Auszutauschende Daten werden in das JSON-Format gebracht und es werden sogenannte JSON-Objekte erzeugt. Das Generieren und Parsen solcher JSON-Objekte ist generell unabhängig von der Programmiersprache, die Sprache benötigt lediglich einen Parser. Daher sind diese JSON-Objekte gut geeignet, um Daten über Netzwerke zu übertragen, da sie von vielen Anwendungen interpretiert werden können. [JSON, 2018, INTRO], [IETF, 2017, RFC8259]

ORM: ORM (*object-relational mapping*) ist eine Technik in der Softwareentwicklung. Sie ermöglicht es Anwendungen, die in einer objektorientierten Programmiersprache geschrieben wurden, ihre Objekte in relationalen Datenbanken abzulegen. Der Anwendung erscheint die relationale Datenbank als objektorientierte Datenbank. In den meisten objektorientierten Programmiersprachen existieren Standardbibliotheken, um diese Technik zu implementieren. [AGILE, 2018, MAPPING]

2.8 Verwandte Arbeiten

Es konnten lediglich eine wissenschaftliche Arbeit und zwei Softwarelösungen gefunden werden, die dem Thema dieser Arbeit gleichen, beziehungsweise ihm sehr nahekommen.

- Bei der gefundenen wissenschaftlichen Arbeit handelt es sich um ein Paper der *Tenable Network Security* aus dem Jahr 2003. In diesem wird ein Verfahren beschrieben, welches ebenfalls direkt den Strom der *Packets* in einem Netzwerk analysiert. Diese Analyse wird mithilfe von Signaturen betrieben und es können Protokolle wie etwa HTTP oder FTP erkannt werden. Es besteht sogar die nicht genauer beschriebene Möglichkeit, zu erkannten Services

Informationen über deren Schwachstellen zu erhalten. Das Verfahren aus diesem Paper beschreibt die kommerzielle Software mit dem Namen *NeVo*. [TENABLE, 2003, NEVO]

- Die erste der beiden gefundenen Softwarelösungen, ebenfalls von der *Tenable Network Security,* ist eine aktuelle Weiterentwicklung der *NeVo*-Software mit dem Namen *Nessus*. Auch diese Software ist kommerziell. Sie bringt das oben im *NeVo*-Paper beschriebene Verfahren zur praktischen Anwendung. [TENABLE, 2018, NESSUS]

- Die zweite der beiden gefundenen Softwarelösungen ist das *Passive Asset Detection System* von Matt Shelton. Hierbei handelt es sich um eine Open-Source-Software, welche ebenfalls Netzwerkdaten passiv aufzeichnet und signaturbasiert analysiert. Auch hier werden Protokolle und Services erkannt. Jedoch findet bei dieser Softwarelösung kein Abgleich der erkannten Services gegen Schwachstellendatenbanken statt. [PADS, 2018, ABOUT]

Die Mehrheit der heutigen Netzwerk-Tools verfolgt den Ansatz, aktiv nach Netzwerkdaten zu suchen. Hierbei wird, mithilfe eines aktiven Scanners, ein Port- und Netzwerkscan durchgeführt. Mit dieser aktiven Methode ist es einfacher, auswertbare Netzwerkdaten zu generieren als mit der passiven Methode. Durch das aktive Aufspüren lassen sich deutlich mehr Analysedaten generieren und dadurch, dass laufende Services aktiv zu einer Antwort angeregt werden, ist es möglich, innerhalb kürzerer Zeit ein vollständiges Bild aller im Netzwerk laufenden Services zu bekommen. Bei der passiven Methode werden lediglich bereits im Netzwerk befindliche Daten analysiert. Weder werden künstliche Testdaten verwendet, noch Services aktiv angesprochen. Es kann daher mit dieser Methode bedeutend länger dauern, genügend viele Netzwerkdaten zu sammeln. Allerdings bietet die passive Methode gegenüber der aktiven zwei entscheidende Vorteile. Zum einen ist sie politisch weniger sensitiv, da aktive Netzwerkscans unter Umständen als Spionage und Teil eines Angriffs gedeutet werden können. Zum anderen werden für passive Netzwerkscans deutlich weniger Ressourcen verbraucht als für aktive.

3 Systemdesign

In diesem Kapitel wird eine Lösung für die Problemstellung dieser Arbeit konzipiert. Die Problemstellung besteht aus folgenden Teilen:

- Filterung von Anwendungsschichtdaten aus Netzwerkverkehr
- Extraktion von Servicenamen und -versionen aus den gefilterten Anwendungsschichtdaten
- Speicherung der extrahierten Informationen
- Prüfung der gespeicherten Informationen gegen eine Schwachstellendatenbank
- Anzeige der Prüfungsergebnisse

Zunächst werden die Anforderungen an den zu entwickelnden Prototypen analysiert und in Form von funktionalen und nicht-funktionalen Anforderungen erhoben. Auf Basis dieser Erhebungen und der Zielstellung der Arbeit werden anschließend alle für einen Systementwurf nötigen Architekturentscheidungen getroffen. Abschließend wird ein Sytem-entwurf für den Prototypen präsentiert.

3.1 Anforderungsanalyse

Um die funktionalen Anforderungen an den Prototypen zu erhalten, müssen die wichtigsten systeminternen Funktionen und Anwendungsfälle

identifiziert werden. Da jedoch zu erwarten ist, dass der Prototyp wenig Interaktion mit dem Benutzer aufweist, wird an dieser Stelle auf eine detaillierte Darstellung der Anwendungsfälle verzichtet. Vielmehr werden die identifizierten Anwendungsfälle in Abschnitt 3.1.1 kurz aufgezählt.

Die nicht-funktionalen Anforderungen an den Prototypen werden hier bewußt allgemein gehalten. Es werden keine Qualitätsziele definiert oder Metriken für diese ausgearbeitet, um sie messbar zu machen. Dennoch werden die allgemein gehaltenen nicht-funktionalen Anforderungen hier aufgeführt, da sie einen Einfluss auf spätere Architekturentscheidungen haben.

Die funktionalen Anforderung sollen im Folgenden mit der Abkürzung FKTA, die nicht-funktionalen Anforderungen mit der Abkürzung NFKTA durchnummeriert werden.

3.1.1 Funktionale Anforderungen

Folgende Anwendungsfälle wurden identifiziert:

- Der Nutzer lässt sich die aufgezeichneten Daten anzeigen.

- Der Nutzer gleicht die aufgezeichneten Daten gegen eine Schwachstellendatenbank ab, um die erkannten Services auf bekannte Schwachstellen zu untersuchen.

Auf Basis dieser Anwendungsfälle und auf Basis der Problemstellung dieser Arbeit wurden folgende funktionale Anforderungen festgelegt:

- **FKTA 1:** Die zu sammelnden Daten sollen in mehreren Netzwerken gleichzeitig gesammelt werden können, um auch größere Netzwerkstrukturen mit Teilnetzwerken etc. untersuchen zu können.

- **FKTA 2:** Das Sammeln von Daten soll mittels autonomer Einheiten dezentral geschehen, um den Verwaltungsaufwand für das Untersuchen eines einzelnen Netzwerkes möglichst gering zu halten. Die gesammelten Daten sollen aus durchsuchten *Packets* (IP-Paketen) gewonnen werden.

- **FKTA 3:** Die gesammelten Daten sollen gespeichert werden können. Die Speicherung soll in einer Datenbank erfolgen, welche an ein System zur Auswertung der Daten angeschlossen ist. Es sollen nur solche Daten abgespeichert werden, welche aus IP-Paketen gewonnen wurden, die Informationen über Servicename und die Version des Services enthalten, der das IP-Paket versendet hat.

- **FKTA 4:** Die gespeicherten Daten sollen vom Nutzer ausgewertet und ihm dargestellt werden können. Auswerten bedeutet hier, dass der Nutzer die Möglichkeit hat, die extrahierten Services mittels Name und Version gegen eine Schwachstellendatenbank zu prüfen. Bei dieser Prüfung werden eventuell vorhandene Schwachstellen für den jeweiligen Service aufgedeckt. Diese werden von der Schwachstellendatenbank ausgegeben und in der Datenbank aus FKTA 3 abgespeichert. Darstellen bedeutet hier, dass der Nutzer sich das Ergebnis der Prüfung anzeigen lassen kann.

3.1.2 Nicht-funktionale Anforderungen

Folgende nicht-funktionale Anforderungen an den Prototypen wurden festgelegt:

- **NFKTA 1:** (Benutzbarkeit) Dem Nutzer soll eine geeignete Anwendung zur Verfügung gestellt werden, mit welcher er FKTA 4 wahrnehmen kann. Im Rahmen dieser Anwendung soll der Nutzer davor geschützt werden, fehlerhafte Eingaben machen zu können.

- **NFKTA 2:** (Effizienz) Die autonomen Einheiten zum Datensammeln sollen möglichst zeiteffizient arbeiten.

- **NFTKA 3:** (Effizienz) Die Speicherung und Auswertung der gesammelten Daten sollen möglichst zeiteffizient sein.

- **NFKTA 4:** (Funktionalität) Das System soll vollständig und korrekt bezüglich seiner funktionalen Anforderungen arbeiten.

- **NFKTA 5:** (Skalierbarkeit) Das Sammeln der Daten soll skalierbar sein. Das bedeutet, dass es in quasi beliebig vielen Netzwerken gleichzeitig geschehen kann.

3.2 Architekturkonzept

Im Folgenden werden auf Basis der Anforderungsanalyse drei Architekturkonzepte vorgestellt. Jedes dieser drei Konzepte erfüllt alle funktionalen Anforderungen an den zu entwickelnden Prototypen und kann Basis eines Systementwurfes sein. Tabelle 3.1 zeigt alle drei Konzepte zusammengefasst. Diese werden anschließend einzeln vorgestellt und bewertet. Abschließend wird auf Basis dieser Bewertungen eines der Konzepte für den Prototypen ausgewählt und alle nötigen weiteren Architekturentscheidungen für dieses gewählte Konzept getroffen.

Konzeptstufen	**Konzept 1**	**Konzept 2**	**Konzept 3**
Datenerfassung	- dezentral	- dezentral	- dezentral
	Designentscheidungen		
	- zentrale Datenhaltung - zentrale Datenauswertung und -darstellung	- zentrale und dezentrale Datenhaltung - zentrale Datenauswertung und -darstellung	- dezentrale Datenhaltung - dezentrale Datenauswertung und -darstellung
Datenhaltung	- zentral	- zentral und dezentral	- dezentral
Datenauswertung	- zentral	- zentral	- dezentral
Datendarstellung	- zentral	- zentral	- dezentral

Tabelle 3.1: Zusammenfassung der möglichen Architekturkonzepte

Vorab lässt sich bereits festhalten, dass die einzelnen Konzepte alle von einer dezentralen Datenerfassung mittels autonom arbeitender Einheiten ausgehen (FKTA 2). Diese durchsuchen allesamt IP-Pakete und gewinnen aus deren Daten selektiv Informationen (FKTA 3). Ausgehend von dieser gemeinsamen Basis werden bei den verschiedenen Konzepten aber unterschiedliche Designentscheidungen bezüglich Datenhaltung, -auswertung und -darstellung getroffen.

3.2.1 Erläuterung Konzept 1

Das Konzept 1 geht von einem Sensor als Datenerfasser aus. Dieser Sensor wird als sehr leichtgewichtige Komponente entworfen. Er soll die IP-Pakete nur filtern und weder eine Datenhaltung noch Datenauswertung oder -darstellung besitzen. Dies hat zur Folge, dass er in der Lage sein muss, die gesammelten Daten immer sofort zu versenden. Um dies gewährleisten zu können, braucht der Sensor eine Art Puffer für die zu sendenden Daten. Ebenfalls eine Folge dieser Entscheidungen ist, dass das System eine zentrale Datenhaltung bekommt. Zu dieser zentralen Datenhaltung senden alle Sensoren ihre Daten. Es muss also neben den Sensoren eine zweite, zentrale Komponente entwickelt werden. Hierbei bietet es sich an, in diese zweite Komponente die Datenauswertung und -darstellung gleich mit zu integrieren. Sie sind also ebenfalls zentral.

Ein Vorteil einer solchen Lösung ist, dass der zu entwickelnde Sensor sehr leichtgewichtig ist, da er quasi keine Logik enthält. Ein leichtgewichtiger Sensor, der nur filtert und versendet, ist sehr zeiteffizient (NFKTA 2). Bedenkt man, dass die passiven Datenerfasser an neuralgischen Punkten in den Netzwerken arbeiten müssen, ist ein effizienter Sensor ein großer Vorteil. Ein weiterer Vorteil dieser Lösung ist, dass durch die zentrale Datenhaltung keine Datenkonsistenzprobleme zu erwarten sind. Dass der Nutzer durch die zentrale Komponente immer alle gesammelten Daten zur Ansicht und Auswertung verfügbar hat, ist ebenfalls ein Vorteil dieser Lösung.

Ein Nachteil dieser Lösung ist, dass zusätzlich zum Sensor eine zweite Komponente entwickelt werden muss. Ein weiterer Nachteil ist, dass sich um die Datenübermittlung zwischen Sensor und zentraler Komponente gekümmert werden muss. Beide brauchen einen Puffermechanismus.

3.2.2 Erläuterung Konzept 2

Das Konzept 2 geht von einer Anwendung aus, die aus zwei Komponenten besteht. Hierbei wird die Datenerfassung mittels einer Anwendung vollzogen. Im Gegensatz zu einem Sensor besitzt diese Anwendung jedoch eine eigene, lokale Datenhaltung. Die Gesamtdatenhaltung in diesem Konzept soll dennoch zentral sein. Die Idee dahinter ist, dass die Anwendung Daten sammelt und lokal speichert. Zu definierten Zeitpunkten synchronisiert sich die Datenbank der Anwendung mit der zentralen Datenbank. Dieses Synchronisieren geschieht nur in Richtung der zentralen Datenbank. Auch hier ist wieder eine zweite, zentrale Komponente zur Datenhaltung zu entwickeln. Ebenfalls bietet es sich hier wieder an, in diese zweite Komponente die Datenauswertung und -darstellung zu integrieren.

Ein Vorteil dieser Lösung ist, dass die lokal gesammelten Daten relativ gut gegen Verlust gesichert sind, was sich mit der Zwischenspeicherung in der lokalen Datenbank begründen lässt. Sie sind, nach der Synchronisation, redundant vorhanden. Ein weiterer Vorteil dieser Lösung ist, dass durch die zentrale Datenhaltung quasi keine Datenkonsistenzprobleme zu erwarten sind. Durch das Synchronisieren ist die zentrale Datenbank fast immer aktuell. Dass der Nutzer durch die zentrale Komponente immer alle, bis zur letzten Synchronisation gesammelten Daten zur Ansicht und Auswertung verfügbar hat, ist ebenfalls ein Vorteil dieser Lösung.

Ein Nachteil dieser Lösung ist, dass die Datensammelkomponente eine Datenbank besitzt. Sie kann somit als mittelgewichtig angesehen werden und ist zeitineffizienter gegenüber einem Sensor, da sie neben dem Filtern der Pakete noch andere Aufgaben hat. Ein weiterer Nachteil dieser Lösung ist, dass sich um das Synchronisieren der Datenbanken gekümmert werden muss. Dies erhöht die Komplexität des Gesamtsystems unnötig. Zudem ist es nachteilig, dass neben den Datenerfassern ebenfalls eine weitere, zentrale Komponente entwickelt werden muss.

3.2.3 Erläuterung Konzept 3

Das Konzept 3 geht von einer aus einer Komponente bestehenden Anwendung aus. Es sieht vor, dass die Datenerfasser jeweils eine lokale Datenhaltung besitzen. Zusätzlich soll es hier keine zentrale Datenhaltung im Gesamtsystem geben. Auch soll das Auswerten und Darstellen der gesammelten Daten nur lokal geschehen. Die zu entwickelnde Anwendung besitzt nur eine Komponente. Diese Komponente kann alle Schritte von der Datenerfassung bis zur Datendarstellung abarbeiten. Sie kann als schwergewichtig angesehen werden, da sie die komplette Anwendungslogik enthält. Das Auswerten und Darstellen der Daten soll nur lokal geschehen. Deshalb muss der Nutzer die Möglichkeit haben, auf die in den einzelnen Netzwerken arbeitenden Anwendungen zuzugreifen. Hierfür brauchen die Anwendungen eine geeignete Schnittstelle. Eine weitere Komponente muss hier nicht entwickelt werden.

Ein Vorteil dieser Lösung ist, dass nur eine Komponente entwickelt werden muss. Ein weiterer Vorteil ist, dass kein Synchronisations- oder Puffermechanismus notwendig ist.

Ein Nachteil dieser Lösung ist, dass die Anwendungen schwergewichtig sind, da sie die gesamte Anwendungslogik enthalten. Ein weiterer Nachteil ist, dass eine zusätzliche Schnittstelle für die Bedienung aus der Ferne entwickelt werden muss und der Benutzer sich in jedes einzelne Netzwerk einloggen muss. Die eingeschränkte Darstellung der gesamten, in mehreren Netzwerken zu sammelnden Daten ist ebenso ein Nachteil, da die Daten eines jeden Netzwerkes einzeln betrachtet werden müssen. Ein weiterer Nachteil ist, dass die gesammelten Daten eines jeden Netzwerkes ebenfalls getrennt ausgewertet werden müssen.

3.2.4 Architekturentscheidungen

Tabelle 3.2 zeigt noch einmal zusammengefasst die Vor- und Nachteile der einzelnen möglichen Architekturkonzepte sowie deren Bewertung. Wie diese Bewertung zustandekommt, wird anschließend erklärt. Nach dieser Erklärung wird eine Entscheidung für eines der Konzepte getroffen. Abschließend werden für dieses Konzept die noch ausstehenden Architekturentscheidungen getroffen.

	Konzept 1	**Konzept 2**	**Konzept 3**
Vorteile	- leichtgewichtige Komponente zur Datenerfassung (zeiteffizient) - keine Daten-konsistenzprobleme - alle gesammelten Daten stets in Gänze einsehbar - alle gesammelten Daten stets in Gänze auswertbar	- gesammelte Daten gut gegen Verlust geschützt da redundant - fast keine Daten-konsistenzprobleme - fast alle gesam-melten Daten stets in Gänze einsehbar - fast alle gesam-melten Daten stets in Gänze auswertbar	- nur eine Kompo-nente zu entwickeln - keine Puffer- oder Synchronisations-probleme zu lösen
Nachteile	- weitere Komponente zu entwickeln - Problem der Daten-übermittlung muss gelöst werden	- weitere Komponente zu entwickeln - mittelgewichtige Komponente zur Datenerfassung (zeitineffizient) - Problem der Synchronisierung mit zentraler Datenhaltung muss gelöst werden	- schwergewichtige Komponente zur Datenerfassung (zeitineffizient) - zusätzliche Schnittstelle zu entwickeln - nur verteilte Sicht auf gesammelte Daten - nur Auswerten der gesammelten Daten für jedes Netzwerk einzeln möglich
Bewertung	**gut**	**ausgewogen**	**nicht gut**

Tabelle 3.2: Bewertung der möglichen Architekturkonzepte

Die Bewertung mit „gut“ für das Konzept 1 lässt sich darauf zurückführen, dass die Vorteile deutlich die Nachteile überwiegen. Zwar müssen zwei Komponenten entwickelt und es muss das Problem der Datenübermittlung

gelöst werden, jedoch überragen die Vorteile bereits mit einem leichtgewichtigen und effizienten Sensor die Nachteile. Die Möglichkeit für den Nutzer, eine zentrale Auswertung und Darstellung der Daten zu erhalten, ist ebenfalls ein großer Vorteil.

Die Bewertung mit „ausgewogen“ für das Konzept 2 lässt sich darauf zurückführen, dass sich Vor- und Nachteile in etwa die Waage halten. Zwar müssen auch hier zwei Komponenten entwickelt werden, jedoch wiegt die Möglichkeit für den Nutzer, ebenfalls wieder eine zentrale Auswertung und Darstellung der Daten zu erhalten, als Vorteil schwer. Die etwas geringere Effizienz des mittelgewichtigen Datensammlers kann gegen den Vorteil aufgewogen werden, dass keine größeren Datenkonsistenzprobleme zu erwarten sind. Als Nachteil ist hier jedoch noch die größere Latenz gegenüber Konzept 1 zu nennen, da nur zu definierten Zeitpunkten synchronisiert wird.

Die Bewertung mit „nicht gut“ für das Konzept 3 lässt sich darauf zurückführen, dass den wenigen Vorteilen viele schwerwiegende Nachteile gegenüberstehen. Zwar ist hier nur eine Komponente zu entwickeln, jedoch wiegt bereits der Nachteil eines schwergewichtigen und damit potentiell ineffizienten Datensammlers schwer. Hinzu kommt, dass nur eine verteilte Auswertung und Darstellung der gesammelten Daten für den Benutzer möglich wäre. Dies ist ein weiterer gravierender Nachteil von Konzept 3. Als Vorteil kann die verminderte Datenübertragung die schweren Nachteile von Konzept 3 jedoch ebenfalls nicht aufwiegen.

Aufgrund der vorgenommenen Bewertungen wurde das Architekturkonzept 1 (Sensor) für den zu entwickelnden Prototypen gewählt. Wie bereits oben angeführt, müssen für dieses Konzept noch weitere Architekturentscheidungen getroffen werden. Diese sind:

I. Wie gewinnt der Datensammler aus den IP-Paketen die interessanten Informationen? Gibt es bereitstehende Lösungen oder muss eine neue Lösung entwickelt werden?

II. Wie wird das Problem der Datenübermittlung zwischen Datenerfasser und der zentralen Datenhaltung gelöst? Gibt es bereitstehende Lösungen oder muss eine neue Lösung entwickelt werden?

III. Welcher Datenbanktyp wird für die zentrale Datenhaltung gewählt?

IV. Wie werden die gesammelten Daten ausgewertet?

V. Welche Schnittstelle wird dem Benutzer zum Auswerten und Darstellen der Daten angeboten?

Die Entscheidungen I. - V. wurden wie folgt getroffen:

I. Zum Durchsuchen von Netzwerkpaketen wurden Sniffer als Lösung ausgemacht. Konkret stellt sich die Frage, ob ein solcher Sniffer selbst entwickelt werden muss oder ob es bereits bestehende Softwarelösungen dafür gibt. Wenn ja, soll das Einbinden einer solchen bereits bestehenden Lösung einer Eigenentwicklung vorgezogen werden.

II. Für das Problem der Datenübermittlung zwischen Sensor und zentraler Datenhaltung wurden Message Broker als Lösungen ausgemacht. Wenn es möglich ist, wird ein solcher bereits bestehender Message Broker genutzt, anstatt selbst einen zu entwickeln. Message Broker verfügen über einen Puffermechanismus, welcher es dem Sensor ermöglicht, die gefilterten Daten sofort zu versenden. Der Sensor braucht keine eigenständige lokale Persistenz und kann leichtgewichtig gehalten werden.

III. Die Datenhaltung erfolgt zentral, daher wird keine verteilte Datenbank benötigt. Die abzuspeichernden Daten werden aus IP-Paketen gewonnen. An dieser Stelle kann noch keine Aussage darüber getroffen werden, ob diese Daten Objekte sind oder nicht. Deshalb wird die Entscheidung für eine relationale Datenbank

getroffen. Sollten die Daten keine Objekte sein, ist die Wahl ohne weiteres gut und richtig. Sollten die Daten Objekte sein, muss an späterer Stelle eine geeignete Lösung, wie etwa ORM, hinzugezogen werden.

IV. Die Auswertung der Daten soll derart erfolgen, dass die gesammelten Daten gegen eine Schwachstellendatenbank geprüft werden. Es gibt prinzipiell zwei Modi, wie eine solche Prüfung stattfinden kann. Zum einen gibt es die Möglichkeit, online eine Prüfung gegen eine Schwachstellendatenbank durchzuführen. Die andere Möglichkeit ist, die gesamte Schwachstellendatenbank aus dem Internet herunterzuladen und die Prüfungen lokal zu machen. Hier wird sich für die zweite Variante entschieden. Begründet wird dies damit, dass die lokale Suche deutlich schneller ist. Erwähnt werden soll noch, dass in einem kommerziellen Umfeld der Sicherheitsaspekt ebenfalls für die zweite Lösung spricht. Er gründet sich auf der Idee, dass bei einer lokalen Suche keine sensiblen Anfragen über das Internet verschickt werden müssen. Selbst wenn diese verschlüsselt verschickt werden, kann ein Angreifer die IP-Adresse mitlesen und herausfinden, dass eine Anfrage an eine Schwachstellendatenbank gestellt wurde. Dies weckt unnötig das Interesse des Angreifers. Ein regelmäßiges, z.B. tägliches Update der lokalen Datenbank kann zwar auch das Interesse eines Angreifers wecken, bietet ihm jedoch weniger Anhaltspunkte, als wenn jede einzelne Anfrage über das Internet verschickt werden müsste.

V. Als Nutzerschnittstelle für das Abgleichen der Daten mit der Schwachstellendatenbank und das Darstellen der Daten wurden zwei Lösungen identifiziert. Entweder wird dem Nutzer eine Anwendung mit einer grafischen Oberfläche (GUI) oder es wird ihm eine Konsolenanwendung zur Verfügung gestellt. Aus Gründen der Einfachheit wurde hier eine Konsolenanwendung gewählt. Diese hat ebenfalls den Vorteil des einfachen Gebrauches der *Manual Pages.*

3.3 Systementwurf

In diesem Abschnitt wird ein Systementwurf für den Prototypen präsentiert. Basierend auf den im vorherigen Kapitel getroffenen Architekturentscheidungen werden drei Sichten auf das zu entwickelnde System gegeben: eine Kontextsicht, eine Bausteinsicht und eine Laufzeitsicht. Die zu den Sichten dargestellten Diagramme sind an die UML-Syntax angelehnt.

3.3.1 Kontextsicht

Abbildung 3.1 zeigt die Schnittstellen zu den Nachbarsystemen und die Interaktionen mit dem Nutzer. Der Prototyp zeichnet Netzwerkverkehr im Zielnetzwerk auf. Der Nutzer kann die gesammelten Daten gegen eine Schwachstellendatenbank prüfen lassen. Außerdem hat der Nutzer die Möglichkeit, sich die gesammelten und ausgewerteten Daten vom Prototypen anzeigen zu lassen. Der Nutzer administriert den Manager und die Sensoren. Der Prototyp lässt, vom Nutzer angestoßen, Daten bei einer Schwachstellendatenbank prüfen und bekommt von dieser das Ergebnis der Prüfung zurück. Er zeigt dem Nutzer abgefragte und ausgewertete, also geprüfte Daten an.

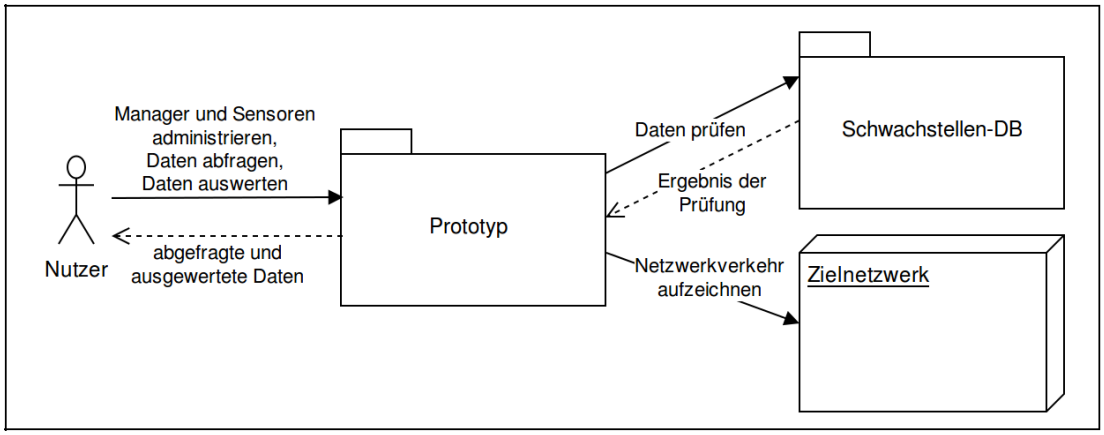

Abbildung 3.1: Kontextsicht des Prototypen

3.3.2 Bausteinsicht

Abbildung 3.2 zeigt die Bausteinsicht auf das Gesamtsystem. Es setzt sich aus zwei Komponenten zusammen. Die Sensoren werden in den einzelnen Netzwerken platziert und zeichnen dort den Netzwerkverkehr auf. Sie filtern bestimmte Informationen aus dem Netzwerkverkehr heraus und senden den gefilterten Netzwerkverkehr zum Manager. Der Manager verwaltet die Daten. Er kann die gesammelten Daten gegen eine Schwachstellendatenbank prüfen und erhält die Ergebnisse der Prüfung von dieser. Weiterhin kann der Manager dem Nutzer abgefragte und ausgewertete Daten ausgeben. Der Nutzer administriert die Sensoren und den Manager und kann Daten abfragen und auswerten lassen.

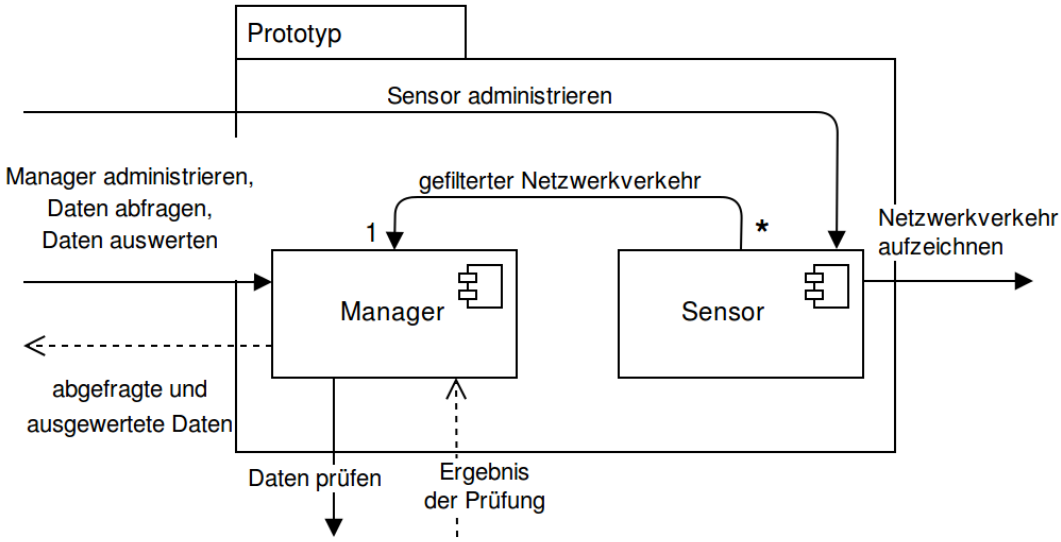

Abbildung 3.2: Bausteinsicht des Gesamtsystems

Abbildung 3.3 zeigt die Bausteinsicht für die Managerkomponente des Prototypen. Diese besteht aus einem Receiver und einem Evaluator als Unterkomponenten und zwei integrierten Datenbanken. Der Receiver empfängt die von den Sensoren gesammelten Daten mittels eines Message Brokers und speichert sie in der RawData-Datenbank ab. Der Evaluator kann die gesammelten Daten von der RawData-Datenbank abfragen und diese in die DetectedServices-Datenbank überführen.

Dieser Vorgang muss vom Nutzer angestoßen werden. Grund für diese Überführung ist, dass in der DetectedServices-Datenbank pro eindeutigem Tupel aus [<Servicename>, <Source-IP>, <Source-MAC>] nur jeweils ein Eintrag stehen soll. Dies ist bei den Daten der RawData-Datenbank nicht der Fall. Weiterhin kann der Evaluator die Daten der DetectedServices-Datenbank zur Prüfung an eine Schwachstellen-datenbank senden und die Ergebnisse dieser Prüfung in der DetectedServices-Datenbank abspeichern. Der Nutzer kann den Manager administrieren. Ebenfalls kann der Nutzer sowohl die Daten aus beiden Datenbanken abfragen und sich somit anzeigen lassen als auch die Daten aus der DetectedServices-Datenbank gegen eine Schwachstellen-datenbank prüfen lassen.

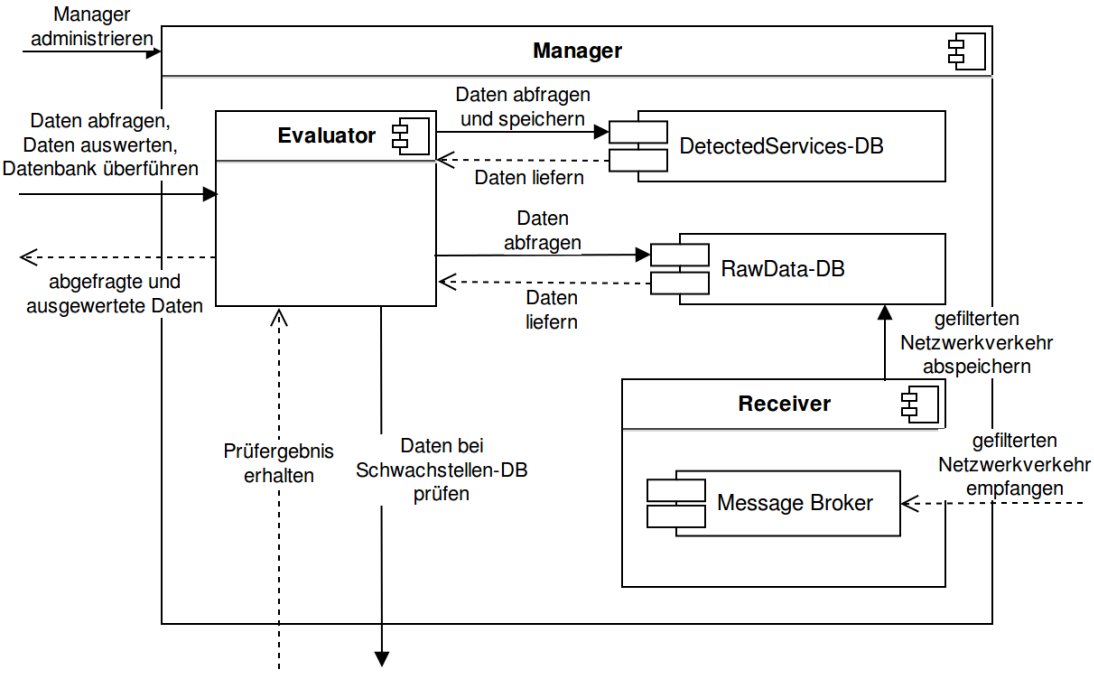

Abbildung 3.3: Bausteinsicht der Managerkomponente

Abbildung 3.4 zeigt die Bausteinsicht für die Sensorkomponente des Prototypen. Ein Sensor zeichnet Netzwerkverkehr mittels eines Sniffers auf. Er filtert diesen nach bestimmten Informationen und überträgt, bei positivem Filterergebnis, die entsprechenden Daten an einen Message Broker. Dieser Message Broker sendet den gefilterten Netzwerkverkehr

anschließend über ein Netzwerk zur Managerkomponente. Der Nutzer kann den Sensor administrieren.

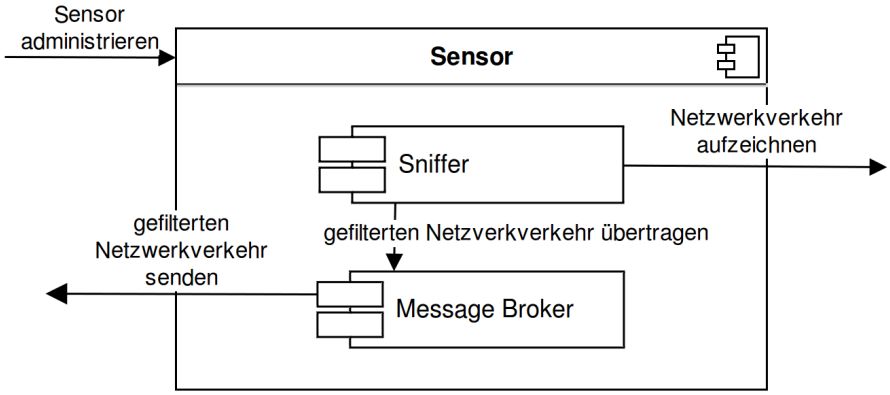

Abbildung 3.4: Bausteinsicht der Sensorkomponente

3.3.3 Laufzeitsicht

Abbildung 3.5 zeigt die Laufzeitsicht des Prototypen. Zunächst muss der Nutzer die einzelnen Sensoren und den Receiver starten. Anschließend hat er mehrere Aktionsmöglichkeiten zur Auswahl. Diese Aktionsmöglichkeiten sind in Abbildung 3.5 mit 1-4 (jeweils umrundet) dargestellt. Der Nutzer kann diese kombiniert, wie unten dargestellt, oder einzeln tätigen. Die einzelnen Aktionen können jeweils beliebig oft wiederholt und auch beliebig miteinander kombiniert werden. Der Nutzer kann sich die gesammelten Rohdaten der Sensoren ansehen. Diese befinden sich in der RawData-Datenbank. Weiterhin kann er sich die bereits detektierten Services ansehen. Diese befinden sich in der DetectedServices-Datenbank. Der Nutzer hat die Möglichkeit, die Datenbank mit den detektierten Services zu aktualisieren, indem er die Daten aus der RawData-Datenbank in die DetectedServices-Datenbank überträgt. Übertragen meint hier, dass aus den Daten der RawData-Datenbank pro eindeutigem Tupel [`<Servicename>`, `<Source-IP>`, `<Source-MAC>`] nur jeweils ein Eintrag in die DetectedServices-Datenbank übernommen werden soll. Dies ist notwendig, da in den Daten der RawData-Datenbank Duplikatpakete auftauchen, welche alle das gleiche eindeutige Tupel besitzen. Sie stammen alle vom gleichen Service und es

hat keinen Mehrwert, sie alle zu prüfen. Deshalb wird nur jeweils ein Exemplar pro eindeutigem Tupel in die DetectedServices-Datenbank übernommen und somit überprüft. Der Nutzer hat weiterhin die Möglichkeit, sich die Einträge aus der DetectedServices-Datenbank anzeigen zu lassen. Diese kann geprüfte und ungeprüfte Datensätze enthalten. Schließlich hat der Nutzer die Möglichkeit, Datensätze aus der DetectedServices-Datenbank gegen eine Schwachstellendatenbank prüfen zu lassen. Die Ergebnisse dieser Prüfung werden in der DetectedServices-Datenbank abgespeichert und dem Nutzer angezeigt.

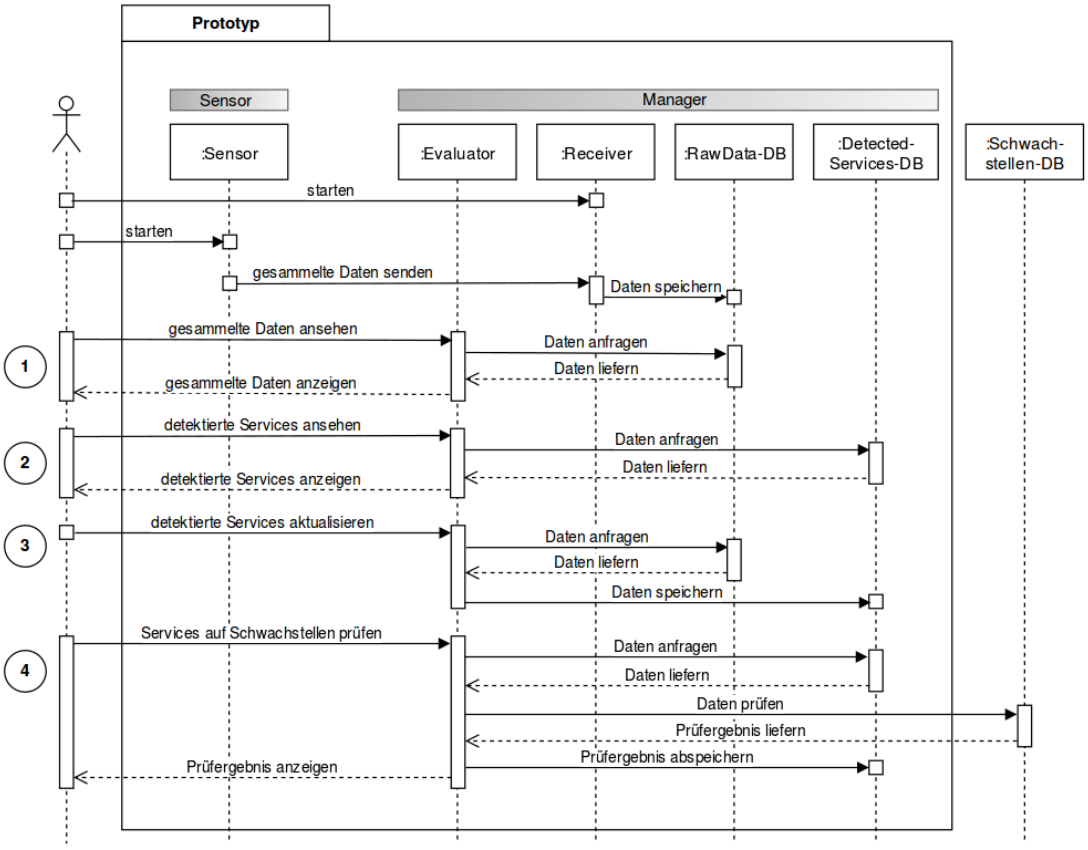

Abbildung 3.5: Laufzeitsicht des Prototypen

4 Implementierung

In diesem Kapitel wird die Implementierung des Prototypen besprochen. Es werden konkrete Technologieentscheidungen gefällt und diese Entscheidungen kurz begründet. Anschließend wird die konkrete Umsetzung der Sensor- und der Managerkomponente gezeigt. Das Kapitel schließt mit der Beschreibung der Testumgebung, mithilfe derer der Prototyp entwickelt und validiert wurde.

4.1 Technologieentscheidungen

An dieser Stelle werden die Entscheidungen über die zur Entwicklung des Prototypen genutzte Programmiersprache und über genutzte Drittsoftware und Bibliotheken getroffen.

4.1.1 Genutzte Programmiersprache

Der Prototyp wird in der Programmiersprache *python* entwickelt. Konkret wird die Version *python 3.5.2* genutzt. Grund hierfür ist der knappe, gut lesbare Programmcode, welcher kompakte und leicht nachvollziehbare Softwareelemente ermöglicht. *Python* bietet dem Entwickler zudem mehrere Programmier-Paradigmen zur Auswahl an, was die

gestalterischen Möglichkeiten erhöht. Die hier entwickelten Komponenten sind allesamt sogenannte Skripte.

4.1.2 Eingebundene Drittsoftware und Bibliotheken

Scapy *(Version 2.3.2)*: *Scapy* ist ein interaktives Programm zur Paketmanipulation und in der Programmiersprache *python* verfasst. Es bedient eine Vielzahl an Netzwerkprotokollen und vereint viele Aufgaben einzelner Netzwerkprogramme in sich. Unter anderem bietet *Scapy* die Möglichkeit, Netzwerkverkehr zu scannen, Pakete zu manipulieren oder Netzwerke zu explorieren. Als eine Unterfunktion bietet *Scapy* einen Sniffer an. Dieser Sniffer wird in der Sensorkomponente des Prototypen zum Aufzeichnen des Netzwerkverkehrs genutzt. Die Entscheidung für *Scapy* wird wegen seiner Vielseitigkeit und wegen des bereitgestellten Sniffers getroffen. [SCAPY, 2018, ABOUT]

Pika: (*Version 0.11.2*) *Pika* ist eine in der Programmiersprache *python* verfasste Bibliothek, welche das AMQP (*Advanced Message Queuing Protocol*) v0.9.1 implementiert. Außerdem bietet *Pika* Erweiterungen für *RabbitMQ*-Clients an. *RabbitMQ* ist ein open-source Message Broker. *Pika* wird im Prototypen eingesetzt, um den in der Sensorkomponente aufgezeichneten Netzwerkverkehr von der Sensorkomponente zur Managerkomponente zu senden und ihn dort zu empfangen. Die Entscheidung für *Pika* mit seinen *RabbitMQ*-Erweiterungen wird getroffen, da es in *python* implementiert, schlank und leicht benutzbar ist. [PIKA, 2018, PROJECT]

cve-search: Als Schwachstellendatenbank mit der größten Verbreitung und wegen ihres Charakters als Industriestandard wurde zum Abgleich der gesammelten Daten die CVE-Datenbank der *Mitre Corporation* gewählt. *cve-search* ermöglicht das lokale Nachschlagen von Schwachstellen in einer CVE-Datenbank. Hierzu importiert *cve-search* die komplette CVE-Datenbank von der *Mitre Corporation* auf eine lokale Maschine. Es besteht zudem jederzeit die Möglichkeit, die lokale Instanz der CVE-Datenbank zu aktualisieren. *cve-search* wird hier eingesetzt, um

die gesammelten Daten des Prototypen gegen eine Schwachstellendatenbank zu prüfen. [CVE-SEARCH, 2018, CVE-SEARCH]

terminaltables: (*Version 3.1.0*) *terminaltables* ermöglicht das Darstellen von Tabellen in Konsolenanwendungen. Der Prototyp soll dem Nutzer Daten auf der Konsole anzeigen können und es ist von Vorteil, wenn diese Darstellung strukturiert und anschaulich ist. *terminaltables* bietet eine gute Möglichkeit dafür an. Es wird im Prototypen genutzt, um die gesammelten und ausgewerteten Daten dem Nutzer auf der Konsole anzuzeigen. [TABLES, 2018, TABLES]

peewee: (*Version 3.1.2*) *peewee* ist ein in *python* geschriebenes ORM (*object-relational mapping*). Es ermöglicht das Abspeichern von Objekten in relationalen Datenbanken. Desweiteren ist *peewee* schlank, einfach und intuitiv zu benutzen. Die hinter *peewee* liegende relationale Datenbank ist eine SQLite-Datenbank. *peewee* wird im Prototypen für die RawData-Datenbank und die DetectedServices-Datenbank in der Managerkomponente eingesetzt. [PEEWEE, 2018, PEEWEE]

argparse: *argparse* ist seit *python 3.2* eine Standard-*python*-Bibliothek. Es handelt sich hierbei um einen Parser für Kommandozeilenbefehle. Die Hauptaufgabe dieser Bibliothek ist es, Befehle, Argumente und Optionen von der Kommandozeile einzulesen und einen Rahmen für entsprechende Reaktionen bereitzustellen. *argparse* ist einfach zu benutzen und wird deshalb gewählt. Im Prototypen wird *argparse* in der Managerkomponente eingesetzt, um die Eingaben des Nutzers in die Konsole zu verarbeiten. [PYTHON, 2018, ARGPARSE]

4.2 Umsetzung der Sensorkomponente

Zunächst werden hier die gemachten Beobachtungen dargelegt. Aus diesen Beobachtungen wird anschließend eine Generalisierung samt Abstraktion hergeleitet. Mithilfe dieser Abstraktion wird die Umsetzung der Sensorkomponente an einem Beispiel konkret gezeigt. Hierbei wird beschrieben, was in dem Prototypen, von der Gewinnung der Daten aus einem IP-Paket bis zu ihrer Anzeige beim Nutzer, geschieht.

4.2.1 Beobachtung

Der von *Scapy* mitgelieferte Sniffer wird gestartet. Mit ihm können Netzwerkpakete mitgelesen und die Ergebnisse auf der Konsole ausgegeben werden. Abbildung 4.1 zeigt beispielhaft die Konsolenausgabe für ein solches mitgelesenes Paket.

```
###[ Ethernet ]###
  dst       = 02:42:ed:2a:d2:4e
  src       = 02:42:ac:11:00:02
  type      = 0x800
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 3577
     id        = 617
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = 6
     chksum    = 0x2c94
     src       = 172.17.0.2
     dst       = 172.17.0.1
     \options   \
###[ TCP ]###
        sport     = http
        dport     = 47954
        seq       = 1531526402
        ack       = 72880064
        dataofs   = 8
        reserved  = 0
        flags     = PA
        window    = 350
        chksum    = 0xbee
        urgptr    = 0
        options   = [('NOP', None), ('NOP', None), ('Timestamp', (7749483,
                    7749482))]

###[ Raw ]###
           load      = 'HTTP/1.1 200 OK\r\nDate: Tue, 13 Mar 2018 17:31:53
           GMT\r\nServer: Apache/2.4.18 (Ubuntu)\r\nLast-Modified: Tue, 13
           Mar 2018 15:19:34 GMT\r\nETag: "2c39-56b28bba3b5ed-
           gzip"\r\nAccept-Ranges: bytes\r\nVary: Accept-
           Encoding\r\nContent-Encoding: gzip\r\nContent-Length:
           3186\r\nKeep-Alive: timeout=5, max=100\r\nConnection: Keep-
           Alive\r\nContent-Type: text/html\r\n\r\n……'
```

Abbildung 4.1: Konsolenausgabe eines mitgelesenen Netzwerkpaketes

Das dargestellte Netzwerkpaket enthält die Datenfelder aus der Sicherungs- (`###[ Ethernet ]###`), der Netzwerk- (`###[ IP ]###`) und der Transportschicht (`###[ TCP ]###`). Zusätzlich sind im dargestellten Paket die Datenfelder der Anwendungsschicht (`###[ Raw ]###`) enthalten. Es ist erkennbar, dass die Quell- und Ziel-Ethernet-Adressen (`###[ Ethernet ]### – dst, src`) sowie auch die Quell- und Ziel-IP-Adressen (`###[ IP ]### – dst, src`) dem Paket entnommen werden können. Auch Quell- und Ziel-Port des Netzwerkpaketes können ihm entnommen werden (`###[ TCP ]### – sport, dport`). Das `load`-Feld der Anwendungsschichtdaten enthält den Namen des Services und seine Version (`Server: Apache/2.4.18(Ubuntu)`) und kann ebenfalls dem dargestellten Paket entnommen werden.

Mit dem Sniffer können zudem Pakete mitgelesen werden, welche jedoch keine Anwendungsschichtdaten (`###[ Raw ]###`) enthalten. Diese liefern zwar alle anderen, oben genannten Informationen, jedoch keine Informationen zum Service, welcher das Netzwerkpaket verschickt hat.

Ebenfalls können mit dem Sniffer Pakete mitgelesen werden, welche zwar Anwendungsschichtdaten enthalten, bei denen es jedoch nicht möglich ist, den Namen und die Version des Services zu entnehmen. Grund hierfür ist, dass diese Informationen nicht im `load`-Feld der Anwendungsschichtdaten enthalten sind.

4.2.2 Generalisierung und Abstraktion

Es wurde beobachtet, dass alle Netzwerkpakete, die vom mitgelieferten Sniffer von *Scapy* aufgezeichnet werden, einer der drei Varianten aus Abschnitt 4.2.1 entsprechen. Es können also folgende allgemeine Aussagen getroffen werden:

- Die Quell- und Ziel-Ethernet-Adressen sind den Paketen immer entnehmbar.
- Die Quell- und Ziel-IP-Adressen sind den Paketen immer entnehmbar.
- Die Quell- und Ziel-Ports sind den Paketen immer entnehmbar.

- Anwendungsschichtdaten sind den Paketen nicht immer entnehmbar.
- Wenn Anwendungsschichtdaten den Paketen entnehmbar sind, bedeutet dies nicht zwangsläufig, dass aus diesen Daten auch der Name und die Version des Services, welcher das Paket verschickt hat, entnehmbar sind.

Aus den Beobachtungen und deren Generalisierung wird nun ein Filtermechanismus erarbeitet, mit dessen Hilfe die mitgelesenen Netzwerkpakete durchsucht werden können. Dies soll hier mithilfe von regulären Ausdrücken geschehen. Bevor diese erarbeitet werden können, muss eine Liste von Services erstellt werden, nach welchen der Sensor die Netzwerkpakete untersucht und die er erkennen soll. Tabelle 4.1 zeigt diese Liste. Auswahl und Zusammenstellung dieser Liste lässt sich mit folgendem Gedankenansatz begründen. Zum einen soll eine gewisse Bandbreite an verschiedenen Services untersucht werden. Hierzu wurden vier verschiedene Arten von Services ausgewählt: Webserver, FTP-Server, Printserver und Datenbankserver. Außerdem sollen zu einer Serviceart mehrere verschiedene Services untersucht werden. Dies ist bei den FTP-Servern der Fall. Zum anderen soll von jedem dieser untersuchten Services jede beliebig verfügbare Version erkannt werden. Zu diesem Zweck wurden für alle Services verschiedene Versionen untersucht. In Tabelle 4.1 ist jeweils nur die neuste untersuchte Version vermerkt.

Service-Art	**Servicename**	**Service-version**	**Standard-Port**	**Protokoll**
Webserver	**Apache2**	2.4.18	80	HTTP
Webserver	**nginx**	1.10.3	80	HTTP
FTP-Server	**vsftpd**	3.0.3	21	FTP
FTP-Server	**proftpd**	1.3.5a	21	FTP
FTP-Server	**pure-ftp**	1.0.36	21	FTP
Printserver	**cups**	2.1.3	631	IPP
Datenbankserver	**mySQL**	5.7.20	3306	mySQL

Tabelle 4.1: Untersuchte Services

Um die regulären Ausdrücke für die in Tabelle 4.1 angegebenen Services zu erarbeiten, müssen die Zeichenketten-Repräsentationen dieser Services aus den jeweiligen `load`-Feldern der Anwendungsschichtdaten in eine verallgemeinerte Form gebracht werden. Beispielsweise hat der Apache-Webserver aus Tabelle 4.1 die konkrete Darstellungsform:

- `Server: Apache/2.4.18(Ubuntu)`

Aus dieser konkreten Darstellung kann mithilfe anderer Apache-Versionen eine verallgemeinerte Darstellung gewonnen werden. Diese lautet:

- `Server: Apache/<Version>(<Betriebssystem>)`

Mithilfe der verallgemeinerten Darstellungsformen werden nun die regulären Ausdrücke zum Filtern der Services erarbeitet. Tabelle 4.2 zeigt die verallgemeinerte Darstellungsform der untersuchten Services sowie den regulären Ausdruck zum Filtern dieser Form aus den `load`-Feldern der Anwendungsschichtdaten.

Service-name	**verallgemeinerte Darstellungsform**	**regulärer Ausdruck**
Apache2	Server: Apache/<Version>(<Betriebssystem>)	Server: [a-zA-Z0-9./]*
nginx	Server: nginx/<Version> (<Betriebssystem>)	Server: [a-zA-Z0-9./]*
vsftpd	220 (vsFTPd <Version>)	220[-a-zA-Z0-9./()]*
proftpd	220 ProFTPD <Version> Server	220[-a-zA-Z0-9./()]*
pure-ftp	N/A	N/A
cups	CUPS/<Version> IPP/<Version>	CUPS [a-zA-Z0-9./]*
mySQL	N/A	N/A

Tabelle 4.2: Abstraktionen zu den untersuchten Services

Als erstes Ergebnis der Untersuchung kann an dieser Stelle bereits festgehalten werden, dass es nicht möglich ist, aus den Anwendungsschichtdaten den Servicenamen oder die -versionen des `pure-ftp`-Services und des `mySQL`-Services zu extrahieren. Daher ist natürlich auch keine verallgemeinerte Form und auch kein regulärer Ausdruck für diese Services verfügbar. Das Kürzel “N/A” aus Tabelle 4.2 meint ebendies.

4.2.3 Umsetzung des Sensors

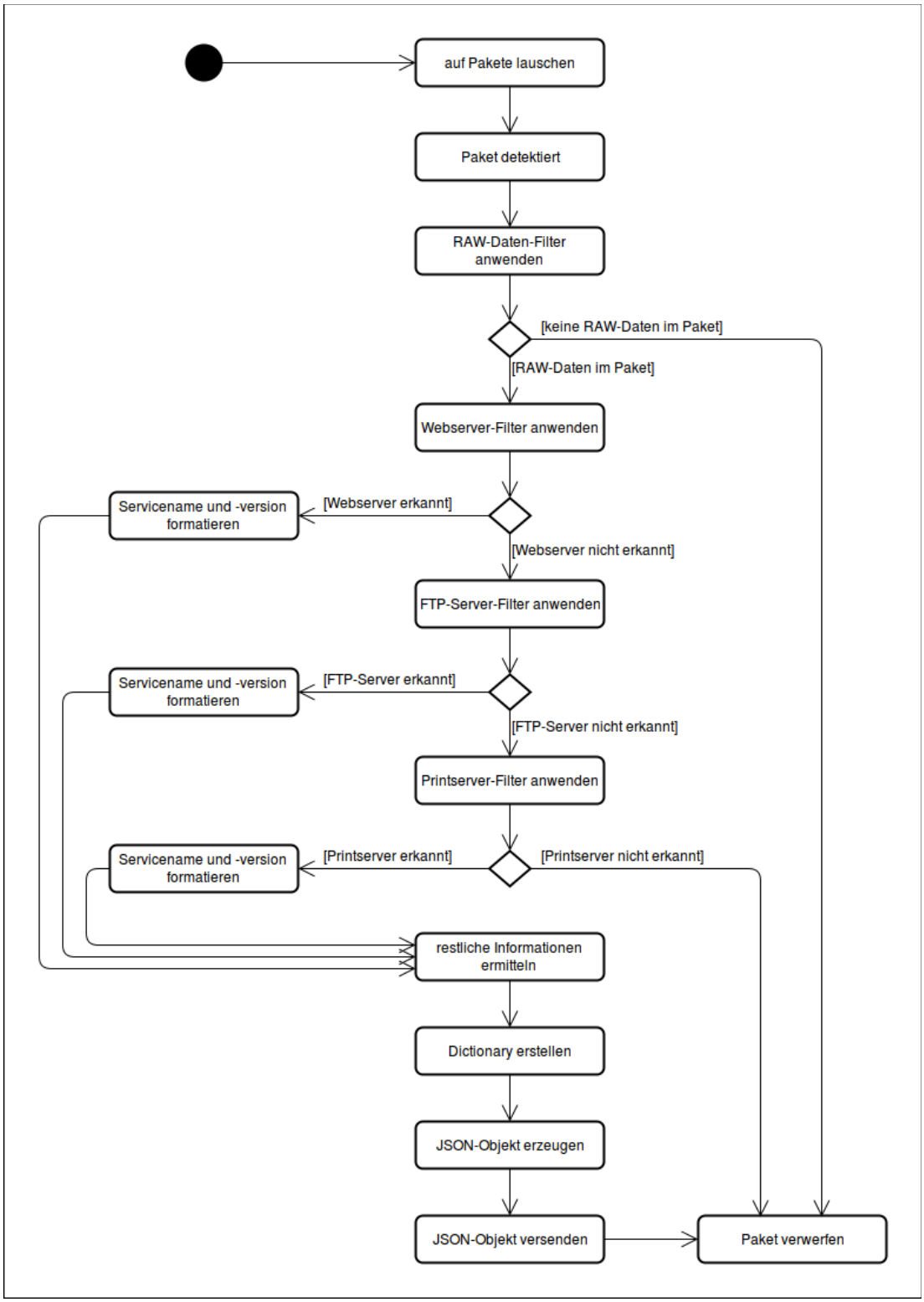

Abbildung 4.2: Funktionsweise der Sensorkomponente

Abbildung 4.2 zeigt die Funktionsweise der Sensorkomponente. Zunächst wird der Sensor vom Nutzer gestartet. Der von *Scapy* mitgelieferte Sniffer ist fest in den Sensor integriert und wird nach dem Start des Sensors automatisch gestartet. Hierzu wird programmintern einmalig die `sniff`-Funktion von *Scapy* aufgerufen. Mithilfe dieser beginnt der Sniffer nun, auf Netzwerkpakete zu lauschen (`auf Pakete lauschen`). Er tut dies auf einem vorher festgelegten Interface. Wird ein Netzwerkpaket von dem Sniffer detektiert (`Paket detektiert`), so ruft die `sniff`-Funktion die `process_paket`-Funktion auf. Diese ist selbst entwickelt und enthält sämtliche weitere Anwendungslogik des Sensors. Abbildung 4.3 zeigt beispielhaft die Funktionsaufrufe beider Funktionen und erläutert deren Parameter.

```
sniff(iface="docker0", prn=process_paket, filter="tcp", store=0)

     #iface  := Interface, auf welchem der Sniffer lauscht
                (z.B docker0 ist das Docker-Interface)
     #prn    := Funktion, welche aufgerufen wird, sobald ein Netzwerkpaket
                detektiert wurde (Funktion bekommt das Netzwerkpaket
                übergeben)
     #filter := Einstellung, auf welche Pakete gelauscht werden soll
                (z.B. "tcp", "udp", "port 80", "port 21", …)
     #store  := Einstellung, ob Paket nach dem Detektieren gespeichert
                werden soll oder nicht (store=0 bedeutet, dass der Sniffer
                das Paket lediglich an die prn-Funktion weiterleitet und
                nicht speichert)
```

```
process_paket(packet)

     #packet := zu verarbeitendes Netzwerkpaket
```

Abbildung 4.3: Funktionsaufrufe der `sniff`- und `process_paket`-Funktion

Weiterhin wird nun, in der `process_paket`-Funktion, das übergebene Netzwerkpaket daraufhin untersucht, ob es Anwendungsschichtdaten enthält. (`RAW-Daten-Filter anwenden`). Ist dies nicht der Fall, wird das Netzwerkpaket verworfen (`Paket verwerfen`) und die `process_paket`-Funktion beendet. Enthält das Netzwerkpaket jedoch Anwendungsschichtdaten, so wird nun mithilfe verschiedener Filter untersucht, ob sich aus diesen die Servicenamen und -versionen der Services aus Tabelle 4.2 extrahieren lassen (Anmerkung: nicht mySQL und pure-ftp). Bei dieser

Untersuchung kommen die, in Abschnitt 4.2.2 erarbeiteten, regulären Ausdrücke zum Einsatz. Sie sind in die jeweiligen Filter integriert.

Zunächst wird versucht, Webserver zu erkennen (`Webserver-Filter anwenden`). Ist dieser Versuch erfolgreich, so werden die jeweils extrahierten Servicenamen und -versionen in ein Format gebracht, welches darauf abgestimmt ist, als Anfrage an eine CVE-Datenbank akzeptiert zu werden (`Servicename und -version formatieren`). Dieses Format besitzt eine spezielle Zeichenkettensyntax. Wird es nicht eingehalten, so akzeptiert die CVE-Datenbank eine Anfrage mit einem nonkonformen Parameter nicht, beziehungsweise findet sie keinen Treffer in ihren Daten, auch wenn die Suchanfrage eigentlich bekannte Schwachstellen ergeben müsste. Der Grund dafür, dass die Daten bereits zu diesem Zeitpunkt CVE-konform formatiert werden, ist folgender: Auf diese Weise können sie, direkt nach ihrem Empfang durch den Receiver der Managerkomponente, in die RawData-Datenbank abgespeichert werden. Sie liegen dann dort bereits in CVE-anfragetauglichem Format vor und müssen von der Managerkomponente nicht weiter bearbeitet werden.

Nachdem die extrahierten Servicenamen und -versionen formatiert wurden, werden die restlichen, für einen Datenbankeintrag relevanten Informationen aus dem zugehörigen Netzwerkpaket ermittelt (`restliche Informationen ermitteln`). Es handelt sich hierbei um folgende Informationen:

- Source-IP-Adresse des Senders des Netzwerkpakets
- Destination-IP-Adresse des Empfängers des Netzwerkpakets
- Source-MAC-Adresse des Senders des Netzwerkpakets
- Destination-MAC-Adresse des Empfängers des Netzwerkpakets
- Inhalt des Last-Modified-Feldes, sofern vorhanden
- HTTP-Statuscode, sofern vorhanden
- Zeitpunkt des Aufzeichnens des Netzwerkpakets

Hierbei ist der Zeitpunkt des Aufzeichnens des Netzwerkpakets nicht direkt aus dem Netzwerkpaket entnehmbar, sondern wird durch eine Unterfunktion vom Programm selbst ermittelt und den Daten hinzugefügt.

Wenn alle relevanten Informationen für einen detektierten Service ermittelt wurden, werden diese anschließend in eine jeweils eigene Map gepackt. Diese Map nennt man in *Python* Dictionary (`Dictionary erstellen`). Nun wird das Dictionary in das JSON-Format gebracht und ein JSON-Objekt daraus erzeugt (`JSON-Objekt erzeugen`). Dieses JSON-Objekt wird schließlich an den integrierten Message Broker übergeben, welcher es an die Managerkomponente versendet (`JSON-Objekt versenden`). Anschließend wird das aktuelle Netzwerkpaket verworfen (`Paket verwerfen`) und die `process_paket`-Funktion beendet. Beim Message Broker handelt es sich um einen *RabbitMQ*-Client, welcher von der *Pika*-Bibliothek zur Verfügung gestellt wird. Die *Pika*-Bibliothek wurde als externe Technolgie in den Sensor eingebunden.

Analog wird bei der Erkennung der FTP-Server (`FTP-Server-Filter anwenden`) und der Printserver (`Printserver-Filter anwenden`) verfahren. Der einzige Unterschied zur oben beschriebenen Erkennung der Webserver ist folgender: Wenn sowohl kein FTP-Server als auch kein Printserver erkannt wurde, wird das Netzwerkpaket direkt nach dem negativen Printserverfilterversuch verworfen, da mit dem Printserverfilter alle integrierten Erkennungsmöglichkeiten aufgebraucht sind.

Die `process_paket`-Funktion wird mit jedem detektierten Netzwerkpaket von der `sniff`-Funktion erneut aufgerufen. Die `sniff`-Funktion selbst läuft in einer Endlosschleife und wird vom Nutzer beendet.

4.3 Umsetzung der Managerkomponente

Die Managerkomponente setzt sich aus dem Reveiver und dem Evaluator, als zwei Unterkomponenten, sowie zwei intergrierten Datenbanken zusammen. Da diese Datenbanken keine Unterkomponenten im eigentlichen Sinne sind, werden die sie betreffenden, relevanten Informationen in den Beschreibungen der Unterkomponenten aufgeführt.

4.3.1 Umsetzung des Receivers

Zunächst wird der Receiver vom Nutzer gestartet. Nun versucht der Receiver, eine Verbindung zu der RawData-Datenbank aufzubauen. Existiert eine solche nicht im selben Verzeichnis, in dem auch das Skript für den Receiver liegt, so legt der Receiver dort ein neues Exemplar einer RawData-Datenbank an. Anschließend wird der Message Broker initialisiert und beginnt, ebenfalls in einer Endlosschleife, auf ankommende Nachrichten von den Sensoren zu lauschen. Kommt eine solche Nachricht von einem Sensor beim Receiver an, geschieht Folgendes: Der Inhalt des Paketes ist ein JSON-Objekt. Der Receiver packt dieses JSON-Objekt wieder aus und verwandelt es zurück in ein Dictionary. Das Dictionary besitzt ebenjene Felder, mit welchen es der Sensor vor dem Versenden versehen hat. Die konkreten Felder eines Dictionaries sind in Abbildung 4.4 dargestellt.

```
dictionary = {'server':         <Servername mit Versionsnummer>,
              'sourceIP':        <sourceIP-Adresse>,
              'destinationIP':   <destinationIP-Adresse>,
              'sourceMAC':       <sourceEthernet-Adresse>,
              'destinationMAC':  <destinationEthernet-Adresse>,
              'lastModified':    <Last-Modified (bei HTTP)>,
              'httpCode':        <HTTP-Statuscode (bei HTTP)>,
              'sniffDate':       <Zeitpunkt der Packetaufzeichnung>}
```

Abbildung 4.4: Felder des vom Receiver empfangenen Dictionaries

Nachdem der Receiver nun Zugriff auf die Felder des Dictionaries hat, liest er diese aus und speichert jeweils ein Dictionary als einzelnen

Datenbankeintrag in der RawData-Datenbank ab. Der Receiver fügt vor diesem Vorgang jedoch noch weitere Felder zu jedem Datenbankeintrag hinzu. Diese weiteren Felder sind die beiden Felder `CVEchecked` und `vulnerability`. Da zu diesem Zeitpunkt noch keine CVE-Prüfung der Services stattgefunden hat, werden diese Felder mit Default-Werten belegt. Abbildung 4.5 zeigt einen Datenbankeintrag des Receivers. Abbildung 4.6 zeigt das Datenbankschema der RawData-Datenbank. Es handelt sich bei der RawData-Datenbank um eine relationale Datenbank, genauer eine SQLite-Datenbank. Um diese bereitzustellen, wurde *peewee* als externe Technologie genutzt.

```
database_entry = {'server':         <Servername mit Versionsnummer>,
                  'sourceIP':        <sourceIP-Adresse>,
                  'destinationIP':   <destinationIP-Adresse>,
                 'sourceMAC':       <sourceEthernet-Adresse>,
                  'destinationMAC':  <destinationEthernet-Adresse>,
                  'lastModified':    <Last-Modified (bei HTTP)>,
                 'httpCode':        <HTTP-Statuscode (bei HTTP)>,
                'sniffDate':       <Zeitpunkt der Packetaufzeichnung>,
                 'CVEchecked':      'no',
                'vulnerability':    ''}
```

Abbildung 4.5: Datenbankeintrag des Receivers

```
class RawDataDB(Model):
  server = CharField()
  sourceIP = CharField()
  destinationIP = CharField()
  sourceMAC = CharField()
  destinationMAC = CharField()
  lastModified = CharField()
  httpCode = CharField()
  sniffDate = DateField()
  CVEchecked = CharField()
  vulnerability = CharField()
```

Abbildung 4.6: Datenbankschema der RawData-Datenbank

Der Grund dafür, dass der Receiver in eine eigene Unterkomponente ausgelagert wurde, liegt darin, dass der Empfang der Nachrichten nicht die eigentliche Arbeit des Evaluators stören soll. Die Anzahl der sendenden Sensoren kann, wie Abbildung 3.2 entnommen werden kann,

größer als eins sein. Damit die ankommenden Nachrichten nicht das Auswerten und Anzeigen des Evaluators stören, wurde die Receiverunterkomponente etabliert.

4.3.2 Umsetzung des Evaluators

Es handelt sich beim Evaluator um das Auswerte- und Anzeigewerkzeug für die von den Sensoren gesammelten Daten. Der Evaluator bietet dem Nutzer eine Konsolenschnittstelle zu seiner Benutzung an. Dies bedeutet, dass der Nutzer für jede Anfrage an den Evaluator einen gesonderten Befehl in die Konsole eintippen muss. Der Evaluator weiß nichts von seiner vorherigen Benutzung. Daher prüft dieser zunächst, ob eine RawData-Datenbank und eine DetectedServices-Datenbank in dem Verzeichnis existieren, in dem auch das Evaluator-Skript liegt. Für die RawData-Datenbank sollte dies der Fall sein, da der Receiver vor dem Evaluator gestartet werden sollte, für die DetectedServices-Datenbank ist dies jedoch nicht klar. Aus diesem Grund legt der Evaluator gegebenenfalls beide Datenbanken im oben genannten Verzeichnis an. Das Datenbankschema der RawData-Datenbank ist Abbildung 4.6 zu entnehmen. Abbildung 4.7 zeigt das Datenbankschema der DetectedServices-Datenbank.

```
class ServiceDB(Model):
  server = CharField()
  sourceIP = CharField()
  sourceMAC = CharField()
  sniffDate = CharField()
  CVEchecked = CharField()
  vulnerability = CharField()
```

Abbildung 4.7: Datenbankschema der DetectedServices-Datenbank

Nun versucht der Evaluator den vom Nutzer eingegebenen Befehl zu parsen. Dies wird mithilfe der *argparse*-Bibliothek bewerkstelligt. *Argparse* wird als externe Technologie in den Evaluator eingebunden. Für Entwickler besteht hier die Möglichkeit, zunächst verschiedene Argumente zu definieren. Nachdem dies geschehen ist, parst der Parser den vom Nutzer eingegebenen Konsolenbefehl. In einem nun folgenden

Abfrageblock wird der geparste Befehl mit allen vorher definierten Argumenten verglichen. Ist der eingegebene Konsolenbefehl für eines der vorher definierten Argumente eine gültige Eingabe, so wird eine entsprechende Reaktion ausgelöst. Diese Reaktion ist vom Entwickler zu bestimmen. Ist der eingegebene Konsolenbefehl für keines der vorher definierten Argumente eine gültige Eingabe, so erfolgt ebenfalls eine vom Entwickler zu bestimmende Reaktion.

Der Evaluator arbeitet exakt nach diesem Schema. Abbildung 4.8 zeigt beispielhaft die Quelltextzeile zur Definition eines solchen Argumentes samt dem dazugehörigen gültigen Konsolenbefehl und der entsprechenden Ausgabe des Evaluators auf der Konsole. Um die Ergebnisse trotz der Konsolenausgabe ansprechend und übersichtlich präsentieren zu können, werden sie in Tabellenform ausgegeben. Um dies zu ermöglichen, wurde *terminaltables* als externe Technolgie eingebunden. Mithilfe von *terminaltables* können Konsolenausgaben in Tabellenform getätigt werden.

Argumentdefinition:

```
parser.add_argument("-sf", "--showfirst", type=int,
                    help="[SHOWFIRST == <number>] shows first <number>
                    entries of database")


                   #-sf         := optionales Argument
                   #--showfirst := andere Repräsentation des optionalen
                                   Argumentes (programmintern genutzt in
                                   dem folgenden Abfrage-Block)
                   #type=int    := positionales Argument (bedeutet hier,
                                   dass hinter -sf in der Befehlseingabe
                                   eine Integerzahl folgen muss)
                   #help="..."  := Hilfetext für das Argument (erscheint
                                   bei falscher Argumentbenutzung, sowie
                                   beim „-help"-Befehl des Programmes
                                   in der Manual Page)
```

getätigter Konsolenbefehl:

```
<Evaluator-Skript-Verzeichnis> $ python3 evaluator.py -sf 1


                   #Bedeutung  := es werden die ersten 1 Einträge aus der
                                  DetectedServices-Datenbank auf der
                                  Konsole ausgegeben
```

entsprechende Konsolenausgabe:

```
+----------+--------------------------+-----------+------------------+----------------------------+------------+---------------+
| Position | server                   | srcIP     | srcMAC           | date                       | CVEchecked | vulnerability |
+----------+--------------------------+-----------+------------------+----------------------------+------------+---------------+
| 1        | apache:http_server:2.4.18 | 172.17.0.2 | 02:42:ac:11:00:02 | 2018-03-13 17:31:53.586796 | yes        | CVE-2017-9798 |
|          |                          |           |                  |                            |            | CVE-2017-7679 |
|          |                          |           |                  |                            |            | CVE-2017-3169 |
|          |                          |           |                  |                            |            | CVE-2017-9788 |
|          |                          |           |                  |                            |            | CVE-2017-7668 |
|          |                          |           |                  |                            |            | CVE-2017-3167 |
|          |                          |           |                  |                            |            | CVE-2016-8743 |
|          |                          |           |                  |                            |            | CVE-2016-8740 |
|          |                          |           |                  |                            |            | CVE-2016-1546 |
|          |                          |           |                  |                            |            | CVE-2016-4979 |
|          |                          |           |                  |                            |            |               |
+----------+--------------------------+-----------+------------------+----------------------------+------------+---------------+
```

Abbildung 4.8: Argumentdefinition, Konsolenbefehl und Konsolenausgabe

Aus Abbildung 4.8 lässt sich entnehmen, dass die auf der Konsole ausgegebene Tabelle stets alle Attributwerte der jeweiligen Datenbankeinträge enthält (vgl. Abbildung 4.7). Lediglich das Attribut „Position" wird in der Tabelle hinzugefügt. Der Wert dieses Attributes entspricht der Position des jeweiligen Eintrages in der DetectedServices-Datenbank.

```
-sa [SA]   #[SA] = ‘packages‘ || ‘services‘,
           #(show all) → 
           #‘packages‘ → zeigt alle detektierten Netzwerk-Pakete,
           #‘services‘ → zeigt alle detektieren Services
-sc        #(show checked) → zeigt alle bereits geprüften Services
-su        #(show unchecked) → zeigt alle noch ungeprüften Services
-sl [N]    #[N] = Integer,
           #(show last) → zeigt die letzten N Einträge der Datenbank
-sf [N]    #[N] = Integer,
           #(show first) → zeigt die ersten N Einträge der Datenbank
-sr [N][M] #[N],[M] = Integer,
           #(show range) → zeigt alle Einträge der Datenbank von
           #Position N bis M
-ss [SS]   #[SS] = String,
           #(show service) → zeigt alle Einträge aus der Datenbank
           #für den Service SS
-sm [SM]   #[SM] = String,
           #(show mac) → zeigt alle Einträge aus der Datenbank für
           #die Ethernet-Adresse SM

-ca        #(check all) → prüft alle Datenbankeinträge gegen die CVE-DB
-cc        #(check checked) → prüft alle bereits geprüften
           #Datenbankeinträge gegen die CVE-DB
-cu        #(check unchecked) → prüft alle ungeprüften Datenbankeinträge
           #gegen die CVE-DB
-cl [N]    #[N] = Integer,
           #(check last) → prüft die letzten N Datenbankeinträge gegen
           #die CVE-DB
-cf [N]    #[N] = Integer,
           #(check first) → prüft die ersten N Datenbankeinträge gegen
           #die CVE-DB
-cr [N][M] #[N],[M] = Integer,
           #(check range) → prüft alle Datenbankeinträge von
           #Position N bis Position M
-cs [CS]   #[CS] = String,
           #(check service) → prüft alle Einträge der Datenbank
           #für den Service CS
-cm [CM]   #[CM] = String,
           #(check mac) → prüft alle Einträge der Datenbank
           #für die Ethernet-Adresse CM

-r         #(refresh) → aktualisiert die detectedServices.db bzgl. der
           #rawData.db

-h         #(help) → zeigt den Hilfetext
```

Abbildung 4.9: Argumente zur Befehlseingabe durch den Nutzer

Grundsätzlich hat der Nutzer vier verschiedene Arten von Befehlen zur Eingabe auf der Konsole zur Auswahl. Abbildung 4.9 zeigt alle für den Evaluator gültigen Konsolenbefehle. Die vier Gruppen von Argumenten

lassen sich ebenfalls aus Abbildung 4.9 entnehmen. Es handelt sich hierbei um:

- Gruppe 1 (Hilfeargument): Dieses Argument (`-h`) sorgt für eine Konsolenausgabe der Manual Page des Evaluators. Zweck ist, dem Nutzer die Hilfetexte der einzelnen Argumente anzuzeigen, damit dieser den Evaluator korrekt benutzen kann.

- Gruppe 2 (Anzeigeargumente): Diese Argumente beginnen sämtlich mit einem `-s` (für „show"). Bei ihrer Benutzung lässt sich der Nutzer lediglich die entsprechenden Daten aus der DetectedServices-Datenbank ausgeben. Es wird nichts geprüft. Die einzige Ausnahme hierbei ist das Argument `-sa packages`. Bei seiner Benutzung werden alle Einträge aus der RawData-Datenbank ausgegeben.

- Gruppe 3 (Prüfargumente): Diese Argumente beginnen sämtlich mit einem `-c` (für „check"). Bei ihrer Benutzung werden die entsprechenden Einträge aus der DetectedServices-Datenbank gegen die CVE-Datenbank geprüft und die Ergebnisse werden dem Nutzer auf der Konsole ausgegeben. Das Schema der angezeigten Tabelle nach einer Prüfung entspricht dem Tabellenschema der Tabelle aus Abbildung 4.8.

- Gruppe 4 (Aktualisierungsargument): Dieses Argument (`-r`) sorgt für eine Aktualisierung der DetectedServices-Datenbank bezüglich der RawData-Datenbank. Was dies genau bedeutet und warum dies geschieht, wird anschließend erläutert.

Grundsätzlich wurde an vorheriger Stelle die Entscheidung getroffen, mit zwei Datenbanken für die aufgezeichneten und zu bearbeitenden Daten zu agieren. Der Grund hierfür stellt sich folgendermaßen dar: Die CVE-Datenbank ist gegenüber dem Prototypen und somit auch gegenüber dem Evaluator ein Fremdsystem. Jede Anfrage des Nutzers auf CVE-Prüfung von Daten aus der DetectedServices-Datenbank wird intern vom Evaluator zu einem eigenständigen Befehl zusammengesetzt. Dieser

Befehl wird dann im Rahmen eines eigenständigen Subprozesses an die CVE-Datenbank abgesetzt. Die CVE-Datenbank führt dann eine Prüfung der übergebenen Daten durch und sendet das Ergebnis dieser Prüfung, ebenfalls im Rahmen des Subprozesses, an den Evaluator zurück. Dieser speichert das Ergebnis in der DetectedServices-Datenbank ab und zeigt es dem Nutzer auf der Konsole an. Die eigentliche Prüfung, welche in der CVE-Datenbank stattfindet, ist jedoch relativ zeitintensiv. Daher soll es unbedingt vermieden werden, Duplikatpakete im Sinne von Abschnitt 3.3.3 zu prüfen. Da jene aber definitiv in der RawData-Datenbank vorliegen, ist eine Duplikatsvermeidung anzustoßen. Dies geschieht durch den Konsolenbefehl aus Abbildung 4.10 (Gruppe 4) und muss vom Nutzer manuell durchgeführt werden. Es wird empfohlen, dies vor jeder Prüfungs- oder Anzeigeaktion oder zumindest zu Beginn jeder Session zu tun. Hierbei wird der aktuelle Stand der RawData-Datenbank duplikatfrei in die DetectedServices-Datenbank überführt. Es werden alle Einträge aus der RawData-Datenbank angeschaut und geprüft, ob ihr eindeutiges Tupel aus `[<Servicename>, <Source-IP>, <Source-MAC>]` bereits in der DetectedServices-Datenbank vorhanden ist. Wenn dies zutrifft, wird der Eintrag nicht übernommen, andernfalls ja. Durch diesen Vorgang der DetectedServices-Datenbank haben neu hinzugefügte Einträge ebenfalls die oben beschriebenen Default-Werte für die Felder `CVEchecked` und `vulnerability` (siehe Abbildung 4.7).

```
<Evaluator-Skript-Verzeichnis> $ python3 evaluator.py -r
```

Abbildung 4.10: Befehl zur Aktualisierung der DetectedServices-Datenbank

Die Entscheidung für zwei Datenbanken liegt nun darin begründet, dass der Nutzer die Möglichkeit haben soll, alle Einträge der Datenbank auf einmal prüfen zu lassen. Dies wäre auch mit einer einzigen Datenbank möglich, jedoch aufgrund der Duplikatpakete wesentlich langsamer bei gleicher Aussagekraft. Dadurch, dass der Nutzer selbst entscheiden kann, wann er die DetectedServices-Datenbank aktualisiert, hat er die Möglichkeit, zeitintensive Prozesse wie oben beschrieben zu minimieren. Trotz dieser Optimierung liegen dem Nutzer zur Ansicht noch alle von den Sensoren aufgezeichneten Netzwerkpakete vor (siehe Gruppe 2 Anzeige-

argumente, `-sa packages`). Generell haben die Einträge aus der RawData-Datenbank mehr Attribute als jene aus der DetectedServices-Datenbank. Hintergrund hierfür ist, dass für eventuelle spätere Auswertungen möglichst viele Informationen zu den aufgezeichneten Netzwerkpaketen gesammelt werden sollten. Für die momentanen Aufgaben des Evaluators (Abbildung 4.9 zu entnehmen) genügen jedoch die Attribute des Datenbankschemas wie in Abbildung 4.7 gezeigt.

Abbildung 4.11 zeigt beispielhaft einen internen Subprozessbefehl, welcher vom Evaluator bei der Prüfung von Daten gegen die CVE-Datenbank getätigt wird. Hierbei wird das *search.py*-Skript des *cve-search*-Programmes aufgerufen. Ihm werden neben Formatierungsparametern (`"-p", "-o"`) der Name des zu prüfenden Services übergeben (`attribute.server`) und es gibt nach seiner internen Prüfung die CVE-IDs des untersuchten Services, soweit vorhanden, an den Subprozess zurück. Diese werden in dem Feld `result` festgehalten und anschließend in die DetectedServices-Datenbank gespeichert.

```
result = subprocess.check_output(["python3",
                                  "<Verzeichnis von cvesearch>/search.py",
                                  "-p",
                                  attribute.server,
                                  "-o",
                                  "cveid"])
```

Abbildung 4.11: Subprozessbefehl des Evaluators an die CVE-Datenbank

Die CVE-Datenbank ist ein Fremdsystem und muss gesondert vom Prototypen aktualisiert werden. Auch muss der Nutzer selbst gewährleisten, dass eine Instanz der CVE-Datenbank vor einer Benutzung der Managerkomponente läuft. Ansonsten würde der Evaluator nicht korrekt arbeiten. Hierzu muss *cve-search* mit dem entsprechenden Befehl gestartet werden.

4.4 Testumgebung

Der Prototyp wurde mithilfe der Docker-Container-Plattform entwickelt und validiert. Hierzu wurde zunächst die Docker-Software installiert. Ist dies geschehen, erscheint bei den Netzwerk-Interfaces des entsprechenden Rechners ein neues Interface. Dies ist das Docker-Interface (`docker0`). Es simuliert einen Zugang zu dem Docker-Netzwerk, welches sich nun auf dem Rechner befindet. Dieses Netzwerk besitzt jedoch noch keine Maschinen. Mithilfe der Docker-Software ist es aber nun möglich, in das simulierte Docker-Netzwerk hinein verschiedene Maschinen samt Betriebssystem zu installieren. Tut man dies, so hat man ein komplett simuliertes Netzwerk, welches sich auf der physikalischen Maschine des eigenen Rechners befindet. Dieses Netzwerk ist nun über das Docker-Interface erreichbar. In die einzelnen Maschinen, welche im Docker-Netzwerk laufen, könnnen nun wie gehabt Services und Programme nachinstalliert werden.

Nach genau diesem Verfahren wurde die Testumgebung für den Prototypen aufgebaut. Es wurden in die einzelnen Maschinen hinein die Services aus Tabelle 4.1 installiert und jeweils gestartet. Gegebenenfalls mussten diese vor ihrem Start noch konfiguriert werden. Nun wurden Anfragen auf diese im Docker-Netzwerk laufenden Services gemacht. Die Anfragen wurden von der physikalischen Maschine aus gemacht, dass heißt von außerhalb des Docker-Netzwerkes. Entweder wurden die Anfragen via Browser, *filezilla* oder *Telnet* gestellt. Die Antworten der einzelnen Services wurden dann ausgewertet. Es war wichtig, dieses Procedere einzuhalten, da nur so passiv aufgezeichneter Netzwerkverkehr erzeugt werden konnte. Es war ebenfalls wichtig, nur die Antworten der Services auszuwerten, da in dieser Arbeit verwundbare Systeme, genauer Services, untersucht werden sollten und keine Clients.

Zum Untersuchen und Auswerten der Antworten wurden zunächst die Services im Docker-Netzwerk gestartet. Dann wurde ein Sensor gestartet. Ihm wurde als Parameter mitgegeben, ausschließlich auf Netzwerkpakete zu lauschen, welche über das Docker-Interface kommen (vgl. Abbildung 4.3). Anschließend wurde der Receiver gestartet, um die Informationen

des Sensors in der RawData-Datenbank speichern zu können. Nun wurden die Anfragen via Browser, *filezilla* oder *Telnet* gemacht. Die Ergebnisse lagen in der Datanbank vor und konnten durch den Evaluator ausgewertet werden. Abbildung 4.12 zeigt beispielhaft je eine Anfrage via *Telnet* und via Browser an einen Webserver, welcher im Docker-Netzwerk läuft.

Anfrage via *Telnet*:

```
$ telnet 172.17.0.2 80
                        #eingetippt in die Kommandozeile der Konsole
                        #172.17.0.2 ist die IP-Adresse der Dockermaschine
                         auf welcher der Webserver läuft
                        #80 ist der Port, auf dem der Webserver läuft
```

Anfrage via Browser:

```
http://172.17.0.2:80
                        #eingetippt in die Browserzeile
                        #172.17.0.2 ist die IP-Adresse der Dockermaschine
                         auf welcher der Webserver läuft
                        #80 ist der Port, auf dem der Webserver läuft
```

Abbildung 4.12: Serviceanfragen via *Telnet* und Browser

5 Ergebnisse

In diesem Kapitel wird zunächst eine kurze Zusammenfassung über die erarbeiten Ergebnisse präsentiert. Anschließend wird ein Ausblick gegeben, welcher für weiterführende Ideen und Arbeiten genutzt werden kann.

5.1 Zusammenfassung

In dieser Arbeit wurde ein Prototyp entwickelt, welcher unverschlüsselten Netzwerkverkehr aufzeichnet und auswertet. Aufgezeichnet wird der Netzwerkverkehr mittels Sensoren in einem oder mehreren Netzwerken. Hierbei werden die Anwendungsschichtdaten der Netzwerkpakete nach Servicenamen und -versionen vorgegebener Services durchsucht. Bei einer erfolgreichen Suche werden die interessanten Informationen aus den Netzwerkpaketen extrahiert und an eine zentrale Komponente gesendet. In dieser zentralen Komponente werden die gesammelten Informationen dann ausgewertet. Sie stellt dem Nutzer hierfür eine Konsolenanwendung zur Verfügung. Auswerten bedeutet konkret, dass die gesammelten Servicenamen und -versionen gegen eine externe CVE-Datenbank geprüft werden. Ziel dieser Prüfung ist es, für die extrahierten Services bekannte Schwachstellen aufzudecken und diese dem Nutzer anzuzeigen. Somit können Netzwerke passiv überwacht werden und

sobald Services mit bekannten Schwachstellen im Netzwerk detektiert werden, können seitens des Nutzers Gegenmaßnahmen eingeleitet werden.

Tabelle 5.1 zeigt die untersuchten Services sowie die Ergebnisse der Untersuchung. Ergebnisse meint hier die Informationen darüber, ob es möglich war, die jeweiligen Servicenamen und -versionen zu extrahieren, nicht die Ergebnisse der jeweiligen CVE-Prüfungen. Für die verschiedenen Services wurden diverse Versionen geprüft. In Tabelle 5.1 sind die jeweils aktuellsten geprüften Versionen verzeichnet, ebenso die Informationen darüber, ob die geprüften Versionen selbst aktuell sind.

Service	Name	Version	Ergebnis
Webserver	**Apache2**	2.4.18 (veraltet)	Name und Version extrahierbar
Webserver	**nginx**	1.10.3 (veraltet)	Name und Version extrahierbar
FTP-Server	**vsftpd**	3.0.3 (aktuell)	Name und Version extrahierbar
FTP-Server	**proftpd**	1.3.5a (veraltet)	Name und Version extrahierbar
FTP-Server	**pure-ftp**	1.0.36 (veraltet)	Name und Version nicht extrahierbar
Printserver	**cups**	2.1.3 (veraltet)	Name und Version extrahierbar
Datenbankserver	**mySQL**	5.7.20 (veraltet)	Name und Version nicht extrahierbar

Tabelle 5.1: Ergebnisse der Untersuchungen

Verglichen mit den vorher definierten Zielen dieser Arbeit (vgl. Abschnitt 1.2) lässt sich festhalten, dass diese umfänglich erreicht wurden. Es konnte unverschlüsselter Netzwerkverkehr aufgezeichnet, aus ihm die interessanten Daten der Anwendungsschicht extrahiert und diese Daten ausgewertet werden. Es wurde ein zufriedenstellender Querschnitt durch die Servicearten der Anwendungsschicht von der Untersuchung abgedeckt.

5.2 Ausblick

Damit dieser Prototyp erfolgreich in einem Monitoring-Kontext eingesetzt werden kann, in dem die Anwendungsschichtdaten von unverschlüsseltem Netzwerkverkehr wie oben beschrieben aufgezeichnet und ausgewertet werden, sind folgende weitere Arbeiten notwendig:

1. Die Breite der untersuchten Services muss erhöht werden.

2. Die in der Kommandozeile verfügbaren Informationen müssen als Input für andere Systeme verfügbar gemacht werden. Hierfür ist eine geeignete Schnittstelle zu entwickeln.

3. Die Konsolenbefehle müssen mitgeloggt werden.

4. Aus den Ergebnissen der CVE-Prüfungen, welche Schwachstellen für einen Service ergeben haben, müssen automatische Warnungen für Administratoren generiert werden, z.B. per E-Mail.

5. Es müssen automatisierte Tests für das System entwickelt werden.

6. Es kann eine GUI entwickelt werden, mit der sich die Konsolenanwendung leichter bedienen lässt. Gegebenenfalls könnte diese GUI in einen Webservice integriert werden, um eine leichtere Bedienung des Systems aus der Ferne zu gewährleisten.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Eckert, 2013] Claudia Eckert. IT-Sicherheit: Konzepte, Verfahren, Protokolle. Oldenbourg Wissenschaftsverlag GmbH, München 2013

[Kappes, 2013] Martin Kappes. Netzwerk- und Datensicherheit: Eine prakische Einführung. Springer Vieweg, Wiesbaden 2013

[EASY-NETWORK, 2018, SNIFFER] Sniffer
https://www.easy-network.de/sniffer.html
(abgerufen am 05.02.18)

[TSPRO, 2018, NETMON] Passive Network Monitoring
http://www.tomsitpro.com/articles/network_monitoring-netflow-it_security-networking-snmp,2-561-2.html
(abgerufen am 22.04.18)

[IANA, 2018, REGISTRY] Port Number Registry
https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml
(abgerufen am 08.02.18)

[IETF, 2000, RFC2828] Request for Comments 2828
https://tools.ietf.org/html/rfc2828
(abgerufen am 08.02.18)

[MITRE, 2018, CVE] CVE List Home
https://cve.mitre.org/cve/
(abgerufen am 07.02.18)

[MITRE, 2018, CVENAME] CVE-2016-1627
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1627
(abgerufen am 07.02.18)

[DOCKER, 2018, WHAT-IS] What is Docker
https://www.docker.com/what-docker
(abgerufen am 09.02.18)

[MANPAGE, 2011, HISTORY] History of UNIX Manpages
https://manpages.bsd.lv/history.html
(abgerufen am 23.04.18)

[3PILLAR, 2018, BROKER] Understanding Message Broker
https://www.3pillarglobal.com/insights/rabbitmq-understanding-message-broker
(abgerufen am 23.04.18)

[JSON, 2018, INTRO] Introducing JSON
https://www.json.org/
(abgerufen am 10.02.18)

[IETF, 2017, RFC8259] Request for Comments 8259
https://tools.ietf.org/html/rfc8259
(abgerufen am 10.02.18)

[AGILE, 2018, MAPPING] Mapping Objects to Relational Databases
http://www.agiledata.org/essays/mappingObjects.html#BasicConcepts
(abgerufen am 11.02.18)

[TENABLE, 2003, NEVO] Passive Vulnerability Scanning
http://www.vodun.org/papers/net-papers/gula_passive_scanning_tenable.pdf
(abgerufen am 23.04.18)

[TENABLE, 2018, NESSUS] Nessus Network Monitor
https://www.tenable.com/products/nessus/nessus-network-monitor
(abgerufen am 23.04.18)

[PADS, 2018, ABOUT] Passive Asset Detection System
http://passive.sourceforge.net/about.php
(abgerufen am 23.04.18)

[SCAPY, 2018, ABOUT] About Scapy
https://scapy.net/
(abgerufen am 20.02.18)

[PIKA, 2018, PROJECT] Project Description
https://pypi.org/project/pika/
(abgerufen am 22.02.18)

[CVE-SEARCH, 2018, CVE-SEARCH] cve-search
https://github.com/cve-search/cve-search
(abgerufen am 24.02.18)

[TABLES, 2018, TABLES] terminaltables
https://robpol86.github.io/terminaltables/
(abgerufen am 24.02.18)

[PEEWEE, 2018, PEEWEE] peewee
http://docs.peewee-orm.com/en/latest/
(abgerufen am 01.03.18)

[PYTHON, 2018, ARGPARSE] argparse
https://docs.python.org/3/library/argparse.html
(abgerufen am 01.03.18)

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____