Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Philipp Gozdzik

Erkennung und Visualisierung von verwundbaren Systemen anhand der Kommunikation zu Update-Services von Linux-Distributionen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Philipp Gozdzik

Erkennung und Visualisierung von verwundbaren Systemen anhand der Kommunikation zu Update-Services von Linux-Distributionen

Bachelorarbeit eingereicht im Rahmen der 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 26.10.2018

Philipp Gozdzik

Thema der Arbeit
Erkennung und Visualisierung von verwundbaren Systemen anhand der Kommunikation zu Update-Services von Linux-Distributionen

Stichworte
IT-Sicherheit, verwundbare Systeme, Sicherheitslücken, Linux, passive Analyse

Kurzzusammenfassung
In dieser Arbeit wird ein Software-Prototyp entworfen und umgesetzt, der den unverschlüsselten Netzwerkverkehr zwischen Linux-Systemen und Update-Services von Linux-Distributionen analysiert, sodass verwundbare Systeme innerhalb eines Netzwerks identifiziert und visualisiert werden können. Hierfür wird ein Verfahren entwickelt, welches Updatekommunikation erkennt und derart auswertet, dass Rückschlüsse darüber gezogen werden können welche Softwarepakete auf den Linux-Systemen eingesetzt werden. Durch den Abgleich mit einer CVE-Datenbank kann auf dieser Basis festgestellt werden, ob diese Softwarepakete über bekannte Sicherheitslücken verfügen.

Philipp Gozdzik

Title of the paper
Detection and visualization of vulnerable systems based on the communication to update services of Linux distributions

Keywords
IT security, vulnerable systems, vulnerabilities, Linux, passive analysis

Abstract
In this thesis, a software prototype is designed and implemented which analyzes unencrypted network traffic between Linux systems and update services of Linux distributions, with the objective to identify and visualize vulnerable systems within a network. For this purpose, a procedure is developed which recognizes update communication and evaluates it to conclude which software packages are installed on the Linux systems. A CVE database can then be used to determine if these software packages have known vulnerabilities.

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation

Mit der fortschreitenden Vernetzung von Diensten und Systemen die tagtäglich in Unternehmen, Organisationen oder im privaten Bereich zum Einsatz kommen, werden immer mehr sensible Daten digital verarbeitet und bereitgestellt. Es bedarf keiner besonderen Erwähnung, dass das Risiko des Zugriffs auf diese Daten durch unbefugte Akteure minimal gehalten werden muss, um Datenverlust und -missbrauch und die daraus entstehenden wirtschaftlichen Konsequenzen zu vermeiden bzw. zu minimieren. IT-Sicherheit nimmt damit immer mehr an Bedeutung zu und stellt IT-Sicherheitsbeauftragte, Softwareentwickler und IT-Administratoren vor große Herausforderungen.

Um dieser Herausforderung gerecht zu werden, kann auf Ebene der Netzwerksicherheit eine Vielzahl von Sicherheitskomponenten eingesetzt werden. Beispielsweise filtern Firewalls den Netzwerkverkehr, sodass nur bestimmte Dienste über die Firewall hinweg kommunizieren können. Als ein weiteres Sicherheitsinstrument analysieren Netzwerk Intrusion Detection Systeme den Netzwerkverkehr auf vom Regelfall abweichende Vorkommnisse, um potentielle Eindringlinge in einem Netzwerk zu entdecken.

Diese Maßnahmen bieten allerdings keinen unmittelbaren Schutz für exponierte Dienste, wie Web- oder Mail-Server. So kann der Webserverdienst eines Unternehmens eine Sicherheitslücke aufweisen, da

zuletzt keine neuesten Updates installiert worden sind. Ein Angreifer kann in diesem Fall versuchen die Verfügbarkeit des Dienstes durch Ausnutzung der Sicherheitslücke zu stören, mit dem Ziel, dem Unternehmen zu schaden oder es gar zu erpressen. Bei schwerwiegenden Sicherheitslücken besteht die Möglichkeit den Host, auf dem der Dienst läuft, zu übernehmen, um so an sensible Informationen zu gelangen oder um von dort aus weitere Angriffe in das Unternehmensnetzwerk zu starten. Aber auch nicht exponierte Systeme, wie Arbeitsplatzrechner, können begünstigt durch vorhandene Sicherheitslücken zu Zielen von Angriffen werden. Erwähnt sei hier der *Adobe Flash Player*, der seit Jahren immer wieder mit äußerst kritischen Sicherheitslücken aufwartet.

Die Überwachung der Aktualität von eingesetzten Softwarekomponenten und existierender Sicherheitslücken ist also ein wichtiger Bestandteil des Maßnahmenkatalogs zur Verbesserung der Sicherheit in IT-Systemen. Sie ist aber ebenso ein aufwändiges Unterfangen, denn es werden jeden Tag neue Sicherheitslücken bekannt. Der Abgleich der von Sicherheitslücken betroffenen Dienste und Anwendungen mit denen, die im eigenen IT-System eingesetzt werden ist zeitkritisch und sollte daher automatisiert überwacht werden, um im Fall von existenten Sicherheitslücken zeitnah reagieren zu können und diese zu beheben.

Zur Identifikation von verwundbaren Systemen können Vulnerability Scanner aktive Netzwerkscans von zu überwachenden Systemen durchführen, um an Informationen über eingesetzte Dienste zu gelangen und eine Überprüfung auf bekannte Sicherheitslücken durchzuführen. Eine weitere Möglichkeit Dienste zu identifizieren, ist die passive Analyse der Kommunikation von im Netzwerk aktiven Diensten, um anhand bekannter Muster, diese zu identifizieren und zu benennen. Ebenso ist es möglich, durch den Einsatz lokaler Überwachungsinstanzen auf den Systemen selbst detaillierte Informationen über installierte Softwarekomponenten zu erhalten, selbst über jene, die nur lokal verwendet werden.

Diese Arbeit will einen weiteren Lösungsansatz entwickeln, bei dem durch die Überwachung des Netzwerkverkehrs Updatekommunikation

identifiziert und ausgewertet wird, um so an Informationen zu eingesetzten Softwarekomponenten zu gelangen und eine Überprüfung auf bekannte Sicherheitslücken zu ermöglichen. Diese Herangehensweise würde es erlauben, die Vorteile von passiver Analyse (keine Einflussnahme auf den Netzwerkbetrieb) mit denen von lokaler Überwachungssoftware (Erfassung von lokalen als auch exponierten Diensten) zu vereinen.

1.2 Ziel und Abgrenzung

Im Fokus dieser Arbeit steht die Entwicklung eines Software-Protoypen, der den Netzwerkverkehr analysiert, sodass verwundbare Systeme im Netzwerk identifiziert werden können. Die Untersuchung des Netzwerkverkehrs durch den Prototypen beschränkt sich hierbei auf die unverschlüsselte Updatekommunikation von Linux-Systemen, die mit einer der gängigen Linux-Distributionen betrieben werden.

Die Architektur des Prototypen soll es erlauben, Systeminformationen dezentral zu sammeln und zentral auszuwerten, damit beispielsweise auch die Überwachung größerer Netzwerke realisierbar ist. Für die zentrale Auswertungskomponente soll eine Benutzerschnittstelle in Form einer Kommandozeilenanwendung angeboten werden. Damit auch eine automatisierte Weiterverarbeitung der gesammelten Daten möglich ist, soll die zentrale Komponente ebenso über eine Programmierschnittstelle verfügen, über die diese Daten in einem geeigneten Format abrufbar sind.

1.3 Zielgruppe

Diese Arbeit richtet sich an alle Personen, die Interesse am Bereich der IT-Sicherheit haben und über grundlegende Informatikkenntnisse verfügen. Dies können beispielsweise Softwareentwickler, IT-Administratoren, IT-Sicherheitsbeauftragte, Informatik-Studenten oder Informatik-Lehrende sein.

1.4 Aufbau der Arbeit

In *Kapitel 2* werden die benötigten Grundlagen behandelt, um den Einstieg in diese Arbeit zu erleichtern. Hierbei werden unter anderem wichtige Industriestandards aus dem Bereich der IT-Sicherheit eingeführt, die für die automatisierte Verarbeitung von Informationen zu Sicherheitslücken und der Zuordnung zu Softwareprodukten essentiell sind.

In *Kapitel 3* werden die Anforderungen an den zu entwickelnden Prototypen aufgestellt und notwendige Kernfunktionen formuliert.

In *Kapitel 4* werden anhand der zuvor definierten Anforderungen die wesentlichen Architekturentscheidungen beschrieben und ein geeignetes System entworfen. Anhand verschiedener Sichten auf das System werden unter anderem Schnittstellen des Systems, Interaktionen mit dem Benutzer und das Zusammenspiel der Komponenten veranschaulicht.

Kapitel 5 beschreibt schließlich die Umsetzung des Prototypen und die dabei eingesetzten Technologien. Wichtiger Bestandteil dieses Kapitels ist die Untersuchung der Updatekommunikation von Linux-Systemen und die Entwicklung eines geeigneten Verfahrens zur Erkennung von angeforderten Softwarepaketen, um so Rückschlüsse auf eingesetzte Software zu ermöglichen.

Kapitel 6 befasst sich mit der Evaluation des umgesetzten Prototypen. Hierbei wird der Prototyp in einer Testumgebung eingesetzt und ein festgelegtes Szenario durchgespielt. Die durch den Prototypen gesammelten Daten werden im Anschluss anhand verschiedener, vorab definierter Gesichtspunkte betrachtet und mögliche Probleme der Umsetzung festgehalten. Des Weiteren werden Vorschläge zur Optimierung gemacht.

In *Kapitel 7* werden die erarbeiteten Ergebnisse dieser Arbeit zusammengefasst und mit der zu Beginn festgelegten Zielsetzung abgeglichen. Zudem werden weitere Ansatzpunkte für aufbauende Arbeiten gegeben.

2. Grundlagen

Dieses Kapitel führt insbesondere in die Grundlagen und Industriestandards aus dem Bereich der IT-Sicherheit ein, die für die automatisierte Verarbeitung von Informationen zu Sicherheitslücken und der Zuordnung zu Softwareprodukten essentiell sind.
Im Speziellen wird hierbei auf die Definition und Abgrenzung zwischen Sicherheitslücken und Schwachstellen *(Abschnitt 2.3)* näher eingegangen. Des Weiteren werden der Standard zur Benennung von Sicherheitslücken CVE *(Abschnitt 2.4)* sowie der CPE-Standard *(Abschnitt 2.5)* zur einheitlichen Benennung von IT-Systemen, Plattformen und Softwarepaketen erläutert. Schließlich werden die verschiedenen Arten von Vulnerability Scannern *(Abschnitt 2.6)* besprochen. Wie mit Hilfe von sogenannten Sniffern der Netzwerkverkehr für weitere Analysen mitgeschnitten werden kann, wird im *Abschnitt 2.7* dargestellt. Den Abschluss des Kapitels bildet die Betrachtung von verwandten Lösungsansätzen in Literatur und in der Praxis.
Es wird davon ausgegangen, dass ein grundlegendes Verständnis über technologische Aspekte wie reguläre Ausdrücke oder das HTTP-Protokoll vorhanden ist. Die Herleitung und Umsetzung eines geeigneten Verfahrens zur Identifizierung und Evaluation von Updatekommunikation beruht im Wesentlichen auf diesen beiden Themengebieten und wird in *Kapitel 4 (Implementierung)* näher beleuchtet.

2.1 Linux-Distributionen

Diese Arbeit findet Anwendung in der Überwachung und Auswertung der Updatekommunikation von Linux-Systemen, die mit unterschiedlichen Linux-Distributionen betrieben werden. Daher soll an dieser Stelle kurz auf diese eingegangen werden.

Der Quellcode von Linux ist frei verfügbar und verwendbar. So ist es möglich ein Linux-System nach spezifischen Anforderungen zusammenzustellen und als Distribution zur Verfügung zu stellen.
Neben dem Linux-Kernel basiert eine solche Distribution meist auf einer Vielzahl von weiteren freien Anwendungen. Wenn also zum Beispiel eine grafische Benutzeroberfläche benötigt wird, können hierfür freie Anwendungen wie KDE oder GNOME zur Distribution hinzugefügt und konfiguriert werden.
Im Laufe der Zeit sind so zahlreiche Distributionen mit verschiedenen Schwerpunkten entstanden. Beispielsweise kann es erforderlich sein, dass eine Distribution nur wenig Speicherplatz belegen darf, um von einem USB-Stick betrieben werden zu können. Andere Distributionen können wiederum an spezielle Hardware- oder Sicherheitsanforderungen angepasst sein. [vgl. Ehses et al., 2012, S.293; Erben, 2017, S.21; Wolfinger, 2013, S.15]

1. Manjaro	*6. MX Linux*
2. Mint	*7. Solus*
3. Ubuntu	*8. Fedora*
4. elementary	*9. openSUSE*
5. Debian	*10. Antergos*

Abbildung 2.1: Überblick über die zehn gefragtesten Linux-Distributionen der vergangenen 12 Monate, erhoben durch DistroWatch durch das sogenannte „DistroWatch Page Hit Ranking“ [vgl. DISTROWATCH, 2018]

Durch die hohe Flexibilität von Linux ergibt sich ein breites Spektrum an Einsatzmöglichkeiten in den unterschiedlichsten Branchen. Die Gemeinsamkeiten unterschiedlicher Linux-Distributionen wiederum könnten nützlich sein, um ein möglichst allgemeines Verfahren zur Identifizierung verwundbarer Linux-Systeme zu entwickeln.

2.2 IT-Sicherheit

Der in dieser Arbeit zu entwickelnde Prototyp und die hergeleiteten Verfahren sollen im Wesentlichen bei der Durchsetzung von Schutzzielen der IT-Sicherheit unterstützen.

Ziel von IT-Sicherheit ist es, den Schutz von Informationen und IT-Systemen zu gewährleisten. Dabei ist eine Untergliederung in eine Vielzahl von Schutzzielen möglich. Die primären Schutzziele der IT-Sicherheit sind:

- *Vertraulichkeit*: Schutz von Informationen vor Zugriff durch unbefugte Dritte

- *Integrität*: Schutz vor Manipulation von Informationen durch unbefugte Dritte

- *Verfügbarkeit*: Gewährleistung der Erreichbarkeit von Diensten und Information für befugte Benutzer

[vgl. Rohr, 2015, S.29]

Die Durchsetzung der Schutzziele bedarf des Einsatzes zahlreicher Sicherheitsmaßnahmen. Die Überwachung der Systeme auf Softwareprodukte mit Sicherheitslücken, so wie es der hier zu entwickelnde Prototyp zum Ziel hat, ist eine mögliche Sicherheitsmaßnahme die hierfür ergriffen werden kann. Der Systemadministrator kann bei identifizierten Sicherheitslücken reagieren und betroffene Systeme patchen, bevor ein Angreifer diese ausnutzen kann.

Die Schutzziele können aber auch durch unzureichende Sicherheitsmaßnahmen verletzt werden. So kann beispielsweise das Vernachlässigen der Durchführung regelmäßiger Updates dazu führen, dass ein Angreifer durch Ausnutzung einer schwerwiegenden Sicherheitslücke einer ungepatchten Anwendung Zugriff auf ein System

erlangt. Wenn er dadurch Zugang zu sensiblen Daten erhält und diese verändert, sind die Schutzziele *Vertraulichkeit* und *Integrität* verletzt. Wenn er wichtige Daten löscht, verschlüsselt oder Dienste herunterfährt, ist die *Verfügbarkeit* dieser nicht mehr gewährleistet. *[vgl. Kappes, 2013, S.4]*

2.3 Sicherheitslücken und Schwachstellen

In der IT-Sicherheit und bei der Durchsetzung von Schutzzielen spielt die Behebung von Sicherheitsproblemen in IT-Systemen eine essentielle Rolle. Um ein Problem zu identifizieren, muss dieses zunächst benannt werden. Dabei wird oftmals nahezu analog von Sicherheitslücken und Schwachstellen gesprochen. Die Differenzierung beider Begriffe ist auch für diese Arbeit wichtig, da sie sich explizit auf die Erkennung von Software mit Sicherheitslücken konzentriert und sich hierbei auf den CVE-Standard (Common Vulnerabilities and Exposures) stützt, welcher im Anschluss beschrieben wird.

Während es im Englischen eine klare Unterscheidung zwischen den Begriffen „Schwachstelle“ (engl. Weakness) und Sicherheitslücke (engl. Vulnerability) gibt, werden im Deutschen beide Begriffe häufig als Synonym verstanden. Nach Rohr ist aber jene Differenzierung besonders wichtig, um ein Sicherheitsproblem genau beschreiben zu können. *[vgl. Rohr, 2015, S.45]*

Er grenzt beide Begriffe wie folgt voneinander ab:

„Eine *Schwachstelle* (auch: „Schwäche“) stellt eine Eigenschaft in der Implementierung, Architektur, Konfiguration oder eines Prozesses dar, die unter bestimmten Bedingungen zu einer Sicherheitslücke führen kann. Eine Schwachstelle ist somit nicht auf den Quelltext einer Anwendung beschränkt, sondern lässt sich auch in der Architektur, der Konfiguration und sogar in organisatorischen Prozessen wiederfinden.“ *[Rohr, 2015, S.45]*

„Eine *Sicherheitslücke* (auch „Verwundbarkeit“ oder „Angreifbarkeit“) bezeichnet das konkrete Auftreten einer oder mehrerer Schwachstellen, über welche die Sicherheit einer Anwendung nachweislich beeinträchtigt werden kann. Einer Sicherheitslücke liegt somit immer mindestens eine Schwachstelle zugrunde.“ *[Rohr, 2015, S.45]*

2.4 CVE

Common Vulnerabilities and Exposures (CVE) ist der Industriestandard zur einheitlichen Bezeichnung und Identifikation von Sicherheitslücken in Standardsoftware. Der Standard vereinfacht den Austausch und die Bereitstellung von Informationen zu einer Sicherheitslücke und bietet eine Grundlage für die Anwendung einer automatisierten Evaluation und Verarbeitung von öffentlich gemachten Sicherheitslücken.

Die wichtigsten Bestandteile eines CVE-Eintrags zur Beschreibung einer Sicherheitslücke sind die Folgenden:

- Eine der Sicherheitslücke zugewiesene CVE-ID (z.B. CVE-2018-9330 oder CVE-2017-18097)
- Kurze *Beschreibung* der Sicherheitslücke
- Relevante *Referenzen* zu Berichten über die Sicherheitslücke

Ein Beispiel für einen konkreten CVE-Eintrag lässt sich der *Abbildung 2.4* entnehmen.

Die Vergabe von CVE-IDs und die Veröffentlichung gemeldeter Sicherheitslücken erfolgt durch eine der *CVE Numbering Authorities* (CNA), die darüber hinaus auch die Beschreibung verfasst, Referenzen hinzufügt und den vollständigen CVE-Eintrag zur CVE-Liste hinzufügt.
[*vgl. CVE-MITRE, 2018, ABOUT]*

Eingeführt wurde der CVE-Standard im Jahre 1999 durch die Non-Profit-Organisation *MITRE Cooperation*, die seitdem eine Liste mit veröffentlichen CVEs verwaltet. Im Jahr 2005 wurde durch das *National*

Institute of Standards and Technology (NIST) die *National Vulnerability Database (NVD)* ins Leben gerufen, die ebenfalls CVE-Einträge zur Verfügung stellt und sich mit der CVE-Liste der *MITRE Cooperation* synchronisiert. Sie lässt sich allerdings durch zusätzliche Filter wie zum Beispiel Betriebssystem, Hersteller, Produktbezeichnung oder Versionsnummer nach veröffentlichten Sicherheitslücken durchsuchen. *[vgl. CVE-MITRE, 2018, NVD]*

Aber auch als Industriestandard, ist das durch die *MITRE Cooperation* verwaltete CVE-Programm nicht frei von Kritik. So wurde auf der *SOURCE Conference* in Boston bemängelt, dass lediglich 60% aller bekannten Sicherheitslücken durch MITRE erfasst werden. Ebenso wird kritisiert, dass tausende von CVEs erst gar keine CVE-ID zugewiesen bekommen, was zur Folge hat, dass zahlreiche Sicherheitsprodukte die auf CVE-IDs angewiesen sind, auf solche Sicherheitslücken nicht korrekt reagieren können. *[vgl. CSO, 2017, CVE-GAP]*

CVE-ID:
CVE-2018-8885
Description:
screenresolution-mechanism in screen-resolution-extra 0.17.2 does not properly use the PolicyKit D-Bus API, which allows local users to bypass intended access restrictions by leveraging a race condition via a setuid or pkexec process that is mishandled in a PolicyKitService._check_permission call.
References:
- *UBUNTU:USN-3607-1*
- *URL:https://usn.ubuntu.com/3607-1/*

Assigning CNA:
MITRE Corporation
Date Entry Created:
20180321

Abbildung 2.4: Beispiel eines CVE-Eintrags [CVE-MITRE, 2018, CVE-2018-8885]

2.5 CPE

Common Platform Enumeration *(CPE)* ist ein Standard zur strukturierten und einheitlichen Benennung von IT-Systemen, Plattformen und Softwarepaketen. Ein CPE-Name repräsentiert dabei keine konkrete Instanz, sondern immer nur eine Klasse eines Produkts. Die Verwendung einer einheitlichen Namenskonvention ist äußerst nützlich, da somit immer eindeutig definiert werden kann, welches System, welche Hardware oder welche Anwendung verwundbar ist. So führt die NVD *(National Vulnerability Database)* des NIST *(National Institute of Standards and Technology)* für alle CPE-Einträge auch Referenzen zu den jeweiligen CVE-IDs, sofern eine Sicherheitslücke vorliegt. Darauf beruht auch die bereits in *Abschnitt 2.4* erwähnte erweiterte Suche nach Produktattributen innerhalb der CVE-Liste der *NVD*.

```
Vendor: microsoft
Product: internet_explorer
Version: 8.0.6001
Update: beta
```

Abbildung 2.5.1: Attribute eines Software-Produkts

Ausgehend von der *CPE 2.3 Naming Specification*, lässt sich ein Produkt mit den in *Abbildung 2.5.1* festgelegten Attributen, in folgende CPE URI Syntax überführen:

```
cpe:/a:microsoft:internet_explorer:8.0.6001:beta
```

Abbildung 2.5.2: Software-Produkt in CPE 2.3 URI Syntax

[vgl. NIST, 2011, S.1]

Das „a“ steht hierbei für *Anwendung*. Weitere zulässige Werte für dieses Attribut wären „o“ für *Betriebssystem* oder „h“ für *Hardware*. Die grundlegende Struktur eines CPE-Namens ist in *Abbildung 2.5.3* zu sehen.

```
cpe:/{part}:{vendor}:{product}:{version}:{update}:
{edition}:{language}
```

Abbildung 2.5.3: Allgemeine Struktur der CPE 2.3 URI Syntax mit einigen möglichen Attributen

[vgl. Foreman, 2009, S.132]

Zusammengefasst ist es also mit den bis hierher definierten Standards *CPE* und *CVE* möglich, genau zu beschreiben, welche Produktklasse (beschrieben durch *CPE*) von welcher Sicherheitslücke (beschrieben durch *CVE*) betroffen ist. Dieser wichtige Zusammenhang wird auch im Rahmen der Realisierung des Prototypen zu berücksichtigen sein.

2.6 Vulnerability Scanner

Vulnerability Scanner sind Werkzeuge, die zur Untersuchung auf Sicherheitslücken in einzelnen Anwendungen, ganzen Systemen oder Netzwerken eingesetzt werden. Der in dieser Arbeit zu entwickelnde Prototyp kann ebenfalls als eine Art Vulnerability Scanner bezeichnet werden, da er Sicherheitslücken in Linux-Systemen anhand der Analyse der Updatekommunikation identifizieren soll. Mit dem folgenden Spektrum an Definitionen soll grob aufgezeigt werden, auf welche verschiedene Arten ein solcher Scanner seiner Aufgabe nachkommen kann. So kann bei *Vulnerability Scannern* zwischen drei Typen unterschieden werden:

- *Source Code Scanner:* Untersuchung des Quellcodes einer Anwendung auf mögliche Probleme, zum Beispiel im Rahmen von Code-Reviews.

- *Application Scanner:* Untersuchung einer Anwendung auf Probleme, die zur Laufzeit auftreten können. Diese Art des Scanners kommt vor allem dann zum Einsatz, wenn der Zugriff auf den Quellcode nicht möglich ist.

- *System Scanner:* Diese Scanner sind in der Lage, ganze Netzwerke und darin betriebene Systeme auf Sicherheitslücken zu untersuchen und zu überwachen.

[vgl. Gregg/Haines, 2012, S.175-177]

Bei der Aufdeckung von laufenden Diensten und Systemen innerhalb von Netzwerken können *System Scanner* auf verschiedene Ansätze zurückgreifen:

- *Active Scanning:* Scanner, die *aktiv* arbeiten, versuchen durch das Versenden von Nachrichten auf Netzwerkebene eine Reaktion beim Zielsystem zu provozieren, um so an verwertbare Informationen zu gelangen. Ein Nachteil, der mit dieser Methode einhergeht, ist allerdings, dass manche Testroutinen das Zielsystem oder einen von dessen Diensten zum Absturz bringen können. Durch das aktive Anfragen von Diensten wird zudem das Netzwerk zusätzlich belastet.

- *Passive Mapping:* Beim *passiven* Ansatz wird der Netzwerkverkehr analysiert, ohne dass in diesen eingegriffen wird. So können beispielsweise auch Dienste, die auf ungewöhnlichen Ports betrieben werden, aufgedeckt werden. Allerdings können Dienste, die im Zeitraum der Untersuchung keine Pakete senden, auf diese Weise nicht entdeckt werden.

- *Lokale Agenten:* Mit lokalen Agenten, die auf den zu überwachenden Systemen selbst installiert werden und diese überwachen, können äußerst genaue Aussagen darüber getroffen werden, welche Dienste auf den Systemen laufen. Die gesammelten Informationen werden dann in der Regel an einen zentralen Monitoring-Server zur weiteren Auswertung übertragen.

[vgl. Gervasi, 2007, S.1019]

2.7 Sniffer

Damit beispielsweise der zuvor eingeführte *System Scanner* den Netzwerkverkehr überhaupt erst analysieren kann, muss er diesen mitschneiden und aufzeichnen können. Dies geschieht in der Regel durch einen Sniffer, der in Form von Hard- oder Software vorliegen kann. Ein solcher Sniffer versetzt die Netzwerkschnittstelle des Hosts, auf dem er betrieben wird, in den sogennanten promiskuitiven Modus. In diesem Modus werden nicht nur die an den Host adressierten Pakete empfangen und sichtbar, sondern auch diese, die es nicht sind. Alle Daten der mitgeschnittenen Pakete sind, sofern nicht verschlüsselt, sofort einsehbar und auswertbar. *[vgl. Gregg/Haines, 2012, S.177-178]*

2.8 Verwandte Arbeiten

Es konnten keine verwandten wissenschaftlichen Arbeiten ausgemacht werden, die im Bezug zum Thema dieser Arbeit stehen.
Es existieren eine Reihe von IT-Sicherheitslösungen mit Vulnerability Scannern, jedoch arbeiten diese in der Regel nicht *passiv*, so wie es eine Auswertung von Updatekommunikation nahe legen würde. Beispielsweise gibt die Dokumentation von *OpenVAS* [Greenbone, 2017], ein Open-Source Vulnerability Assessment System, keinerlei Hinweise auf eine *passive* Funktionsweise des integrierten Scannermoduls.
Als der einzige auffindbare Vertreter für *passive Vulnerability Scanner* sei hier der *Passive Vulnerability Scanner (PVS)* von *Tenable* erwähnt. Dem zugrundeliegenden Patent [Patent US 7761918 B2, 2010] lässt sich allerdings entnehmen, dass der Scanner wohl keine Updatekommunikation auswertet, da keine Beschreibung zu einem solchen Verfahren formuliert wurde. Ebenso lässt die Zusammenfassung der Funktionen in [Tenable, 2015] den Schluss zu, dass im Wesentlichen nur Anwendungen und Dienste identifiziert werden können, die am Austausch von Daten im Netzwerk partizipieren. Eine Auswertung der Updatekommunikation würde jedoch auch Rückschlüsse auf installierte Anwendungen zulassen, die sonst nur lokal zur Verfügung stehen.

3. Anforderungen

Für die Entwicklung eines Prototypen müssen zunächst die Anforderungen an diesen bestimmt werden, die zur Erreichung der Zielsetzung dieser Arbeit erfüllt werden müssen. Hierzu sollen die in *Abschnitt 1.2 Ziel und Abgrenzung* angestrebten übergeordneten Ziele zur Veranschaulichung nochmal benannt und herangezogen werden:

- Erkennung und Analyse von Updatekommunikation
- Identifizierung von verwundbaren Linux-Systemen
- Dezentrale Aufzeichnung und Analyse von unverschlüsselter Updatekommunikation
- Zentrale Auswertung der aufgezeichneten Daten zur Identifizierung von verwundbaren Linux-Systemen
- Umsetzung einer Benutzerschnittstelle in Form einer Kommandozeilenanwendung
- Umsetzung einer Programmierschnittstelle, über die die gesammelten Daten in einem geeigneten Format abrufbar sind.

Aus den übergeordneten Anforderungen an den Prototypen werden nachfolgend die wesentlichen funktionalen und nicht-funktionalen Anforderungen an das zu entwickelnde System definiert. Für die Ableitung der funktionalen Anforderungen werden zusätzlich einige Anwendungsfälle identifiziert und berücksichtigt.

3.1 Funktionale Anforderungen

Neben den aus der Zielsetzung abgeleiteten Anforderungen können zur Ermittlung der funktionalen Anforderungen an den Prototypen zusätzlich Anwendungsfälle herangezogen werden. Um dem Anwender grundlegende Funktionen zur Konfiguration des Prototypen und der Überwachung eines Netzwerks zur Verfügung zu stellen, konnten Anwendungsfälle identifiziert werden, die nachfolgend in vereinfachter Form formuliert werden. Der hier als „Nutzer“ bezeichnete Akteur kann eine Person sein, die mit der Überwachung eines Netzwerks betraut ist, beispielsweise ein IT-Administrator.

- Der Nutzer konfiguriert den Prototypen, um ihn für die Überwachung des Netzwerks einzurichten.
- Der Nutzer lässt sich die Kommunikation zwischen den zentralen und dezentralen Komponenten anzeigen, um eine fehlerhafte Konfiguration des Prototypen oder des Netzwerks aufdecken zu können.
- Der Nutzer lässt sich alle oder bestimmte im Netzwerk identifizierte Softwarekomponenten anzeigen.
- Der Nutzer veranlasst die Überprüfung von allen oder bestimmten im Netzwerk identifizierten Softwarekomponenten auf bekannte Sicherheitslücken, um somit verwundbare Systeme aufzudecken.
- Der Nutzer lässt sich detaillierte Informationen zu einer Sicherheitslücke anzeigen, die für eine im Netzwerk identifizierte Softwarekomponente gefunden wurde.

Unter Berücksichtigung der Zielsetzung sowie der identifizierten Anwendungsfälle können folgende funktionale Anforderungen (FA) festgesetzt werden:

FA1: Zentrale Datenhaltung und Evaluation von gesammelten Daten

Die Datenhaltung und Prüfung auf Sicherheitslücken für identifizierte Anwendungen und Dienste in einem Netzwerk soll in einer zentralen Anwendung geschehen. Die hierfür benötigten Daten können durch im Netzwerk verteilte Sensoren gesammelt und zugestellt werden.

FA2: Verteilte Sensoren

Ein *Sensor* wird in dieser Arbeit als ein Dienst definiert, der den Netzwerkverkehr mitschneidet, eine Vorfilterung vornimmt und relevante Informationen für das Gesamtsystem extrahiert.
So sollen mehrere Instanzen solcher Sensoren in verschiedenen Bereichen eines Netzwerks platzierbar sein und ihre Ergebnisse an einen zentralen Dienst übermitteln können.

FA3: Konfigurierbare Sensoren

Die Sensoren müssen in einem sinnvollen Maße vom Anwender konfigurierbar sein, um ihrer Aufgabe im Netzwerk nachkommen zu können. Beispielsweise sollten das Interface und der Port, auf dem der Netzwerkverkehr überwacht werden soll, sowie die IP-Adresse der zentralen Auswertungskomponente, an die die Ergebnisse gesendet werden sollen, konfiguriert werden können.

FA4: Identifizierung von Softwarenamen und Versionsnummern

Sensoren sollen anhand des aufgezeichneten Netzwerkverkehrs Update-Anfragen identifizieren und Softwarenamen sowie

Versionsnummern der aktualisierten Softwarepakete extrahieren können.

FA5: Logging von Sensoren

Ein Sensor muss eine Art Ausgabe bereitstellen, mit der sich nachvollziehen lässt, wie der mitgeschnittene Netzwerkverkehr interpretiert wurde. Beispielsweise kann eine mitgeschnittene Anfrage verworfen werden, da sie vom Sensor nicht als Update-Anfrage eingestuft wurde. Eine solche Entscheidung muss vom Nutzer einsehbar sein. Ebenso muss erkennbar sein, wenn Nachrichten an die Auswertungskomponente gesendet werden.

FA6: Logging des Empfängers

Ein Empfänger von Sensor-Nachrichten muss eine Art Ausgabe bereitstellen, mit der sich nachvollziehen lässt, welche Nachrichten von Sensoren empfangen wurden. Beispielsweise um Sensoren zu identifizieren, die ihre Nachrichten aufgrund einer falsch konfigurierten Firewall nicht erfolgreich übertragen können.

FA7: Abgleich identifizierter Anwendungen mit einer lokalen CVE-Datenbank

Die Untersuchung der identifizierten Anwendungen auf existierende Sicherheitslücken muss anhand einer lokal vorliegenden CVE-Datenbank durchgeführt werden. Durch den Abgleich mit einer externen zum Beispiel im Internet lokalisierten CVE-Datenbank besteht sonst das Risiko, dass Dritte in Erfahrung bringen könnten, welche Anwendungen und Dienste in welcher Version im überwachten Netzwerk zum Einsatz kommen. Ein potentieller Angreifer, der über diese Informationen verfügt, kann diese ausnutzen, um seine Chancen auf einen erfolgreichen Angriff auf das Netzwerk zu erhöhen.

FA8: Anzeige der bisher identifizierten Anwendungen

Dem Benutzer soll eine Reihe von Filtern zur Verfügung gestellt werden, mit deren Hilfe er sich einen individuellen Überblick über die bisher entdeckten Anwendungen im Netzwerk verschaffen kann. Es sollte unter anderem erkennbar sein, auf welchem System eine Anwendung identifiziert wurde, ob eine Überprüfung auf Sicherheitslücken bereits stattgefunden hat und ggf. die CVE-ID bei einer entdeckten Sicherheitslücke. Praktische Filter, um auch in größeren Datenmengen die Übersicht zu behalten, könnten folgende sein:

- MAC-Adresse des Systems
- Netzwerk-Bezeichnung (definiert bei Sensor-Konfiguration)
- Nicht geprüfte Einträge
- Geprüfte Einträge
- Einträge mit identifizierten Sicherheitslücken
- Zeitraum
- Bereich von Einträgen
- Anzahl letzter Einträge
- Kombination aus obigen Filtern

FA9: Überprüfung identifizierter Anwendungen auf Sicherheitslücken durch den Benutzer

Der Benutzer soll die Möglichkeiten haben, die Überprüfung beliebiger identifizierter Anwendungen zu veranlassen, auch derer, die bereits überprüft worden sind. Dabei soll wie in FA8 eine Reihe von Filtern zur Auswahl stehen, um die gewünschten Einträge zu selektieren.

FA10: Anzeige detaillierter Informationen zu Schwachstellen anhand betroffener Anwendungen

Für den Fall, dass eine konkrete Anwendung von einer Schwachstelle betroffen ist, soll der Benutzer weitere Informationen über diese einholen können. Aus selbigen Gründen wie in FA7 beschrieben sollte auch hier die Beschaffung der Informationen über eine lokale CVE-Datenbank geschehen.

FA11: Bereitstellung einer Benutzer- und Programmierschnittstelle zur Verwendung der zentralen Anwendung

Die in in FA7-FA11 definierte Funktionalität soll innerhalb der in FA1 definierten zentralen Anwendung mittels einer Benutzer- sowie Programmierschnittstelle verfügbar sein.
Die Benutzerschnittstelle sollte hierbei mindestens durch eine Kommandozeilenanwendung realisiert werden.

3.2 Nicht-funktionale Anforderungen

Die im Folgenden erhobenen nicht-funktionalen Anforderungen (NFA) definieren die qualitativen Merkmale, die der Software-Prototyp vorweisen sollte. Diese werden beim Systementwurf und der Implementation des Prototypen berücksichtigt, jedoch wird in dieser Arbeit darauf verzichtet den Grad ihrer Umsetzung anhand von Metriken zu messen und zu bewerten.

NFA1: Benutzerfreundlichkeit

Benutzer, die bereits erste Erfahrungen mit Kommandozeilenanwendung sammeln konnten, sollten bereits nach kurzer Zeit den vollen Funktionsumfang der zentralen Anwendung erlernen und ausnutzen können.

NFA2: Erweiterbarkeit

Der Prototyp soll derart erweiterbar sein, dass mögliche zukünftige Erweiterungen, wie beispielsweise eine grafische Benutzeroberfläche oder auch andere Anwendungen über eine allgemeine Programmierschnittstelle Zugriff auf die bestehende Funktionalität der zentralen Anwendung haben. Diese Schnittstelle sollte ausreichend dokumentiert sein, um eine transparente Nutzung zu ermöglichen.

NFA3: Funktionalität

Der Prototyp setzt die in den funktionalen Anforderungen definierte Funktionalität vollständig und korrekt um. Wenn eine korrekte Erkennung von verwundbaren Systemen nicht in jedem Fall gewährleistet werden kann, so sind tendenziell falsch-negative Entscheidungen zu vermeiden und falsch-positive vertretbar, sofern diese in überprüfbarer Anzahl auftreten.

4. Systementwurf

In diesem Kapitel werden der entwickelte Systementwurf und die grundlegenden Architekturentscheidungen beschrieben. Zunächst wird das grundlegende Konzept *(Abschnitt 4.1)* für den Prototypen vorgestellt. Anschließend werden anhand der Kontextsicht *(Abschnitt 4.2)* die Schnittstellen zur Nachbarsystemen beschrieben. Die Verteilungssicht in *Abschnitt 4.3* beschreibt die wesentlichen technischen Rahmenbedingungen, in denen der Prototyp zur Ausführung kommt. Schließlich wird mit Hilfe der Bausteinsicht *(Abschnitt 4.4)* der interne Aufbau des Prototypen und seine Architekturbausteine dargestellt. Den Abschluss bildet das Sequenzdiagramm in *Abschnitt 4.5*, welches die Interaktionen zwischen den Modulen des Prototypen veranschaulicht.

4.1 Konzept

Die in *Kapitel 3* definierten funktionalen Anforderungen wirken sich maßgeblich auf den Systementwurf aus. Insbesondere die Anforderung der zentralen Datenhaltung und Auswertung der gesammelten Daten (FA1) sowie die Anforderung der dezentralen Analyse des Netzwerkverkehrs (FA2) führen zum Ansatz, den Prototypen in zwei Subsysteme aufzuteilen.

So ist für die zentrale Datenhaltung und Auswertung der Daten das sogenannte *Überwachungs-System* vorgesehen. Die Daten erhält dieses von den dezentralen *Sensor-Systemen*, die beispielsweise in den Subnetzwerken eines Unternehmensnetzwerks platziert werden und dort den Netzwerkverkehr der umliegenden Systeme analysieren.

Abbildung 4.1 skizziert vereinfacht die Funktionsweise für den zu entwickelnden Prototypen anhand eines beispielhaften Einsatzszenarios in einem Netzwerk mit einer typischen Kompartmentalisierung in Unternetze. Es wird aufgezeigt, wie sich die Subsysteme des Prototypen in den Unternetzen einbetten lassen und miteinander interagieren, um eine Überwachung der Systeme des Netzwerks zu realisieren.

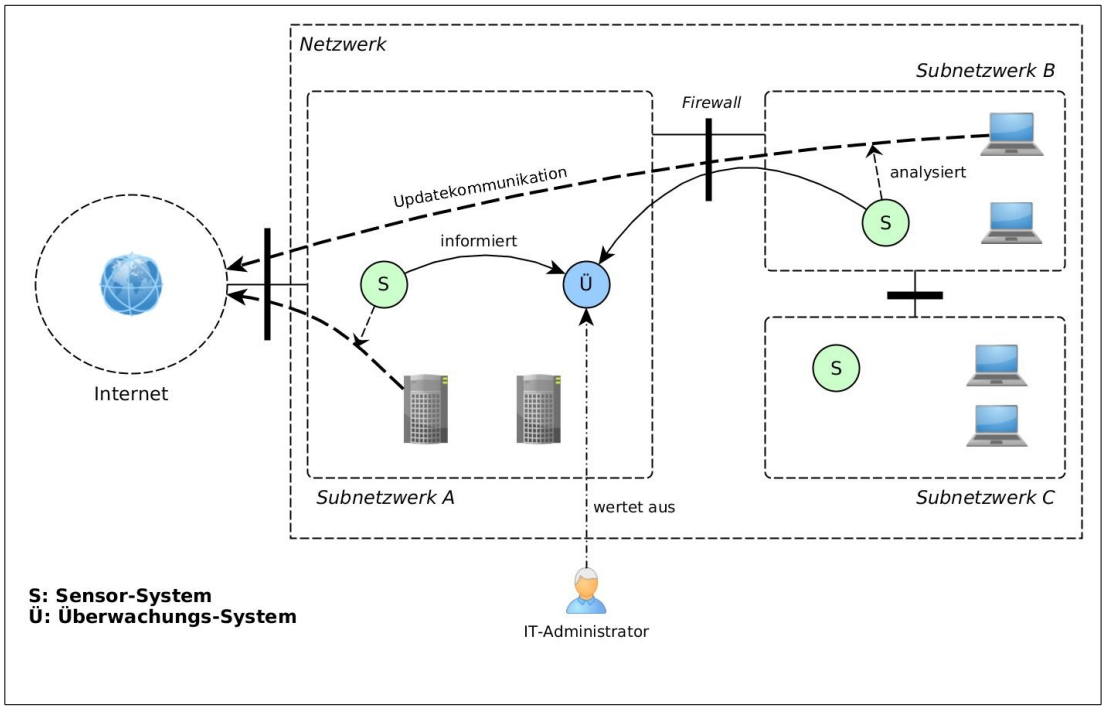

Abbildung 4.1: Skizzierung eines beispielhaften Einsatzszenarios für den Prototypen in einem Netzwerk mit einer Kompartmentalisierung in Unternetze

Das *Sensor-System* ist also ein in einem Netzwerk frei platzierbarer Dienst, der den Netzwerkverkehr mitschneidet, ausgehende Software-Update-Anfragen identifiziert und diese an das zentrale *Überwachungs-System* übermittelt. Mehrere solcher in einem Netzwerk verteilter *Sensor-Systeme* versorgen ein zentrales *Überwachungs-System* mit Informationen über durchgeführte Updates der Linux-Systeme im Netzwerk. Durch den Einsatz von verteilten Sensoren in den Subnetzwerken ist es möglich, Informationen über die Linux-Systeme aus der Innensicht dieser Subnetzwerke zu gewinnen. Damit ist gemeint, dass

beispielsweise auch beim Einsatz von NAT-Komponenten im Netzwerk die eigentlichen IP-Adressen der Linux-Systeme aufgezeichnet werden können.

Das *Überwachungs-System* ist die zentrale Anlaufstelle für den Nutzer, um sich einen Überblick über die Sicherheitslage im gesamten Netzwerk zu verschaffen. Es vereint die Datenhaltung identifizierter Dienste und Anwendungen im gesamten Netzwerk und die Überprüfung auf vorhandene Sicherheitslücken.

Durch die Aufteilung in zwei Subsysteme können die funktionalen Anforderungen FA2 bis FA5 durch das *Sensor-System* und die funktionalen Anforderungen FA1 und FA6 bis FA11 durch das *Überwachungs-System* umgesetzt werden.

4.2 Kontextsicht

Die in *Abbildung 4.2* dargestellte Kontextsicht zeigt die Schnittstellen zu Nachbarsystemen und die möglichen Interaktionen mit dem Benutzer.

Schnittstellen: Die einzige Schnittstelle zu Nachbarsystemen, die der Prototyp benötigt, ist die zu einer Anwendung, die eine durchsuchbare CVE-Datenbank verwaltet. In *Kapitel 5* wird genauer erläutert, auf welche Anwendung hier zurückgegriffen wird.

Benutzer: Es wird davon ausgegangen, dass nur eine Art von Benutzer mit der Verwendung und Betreuung des Systems betraut wird – dies könnte zum Beispiel ein IT-Administrator sein. Die Interaktion des Benutzers mit den *Sensor-Systemen* beschränkt sich lediglich auf das Konfigurieren und Starten dieser. Beim *Überwachungs-System*, dem zentralen Werkzeug zur Auswertung der identifizierten Anwendungen, kann der Benutzer durch eine Kommandozeilenanwendung beispielsweise eine Suche nach Sicherheitslücken anstoßen. Es sei der Vollständigkeit halber erwähnt, dass das Nachbarsystem (die CVE-Datenbank) ebenso

durch den Benutzer auf dem neuesten Stand gehalten werden muss, um die Effektivität des *Überwachungs-Systems* zu gewährleisten.

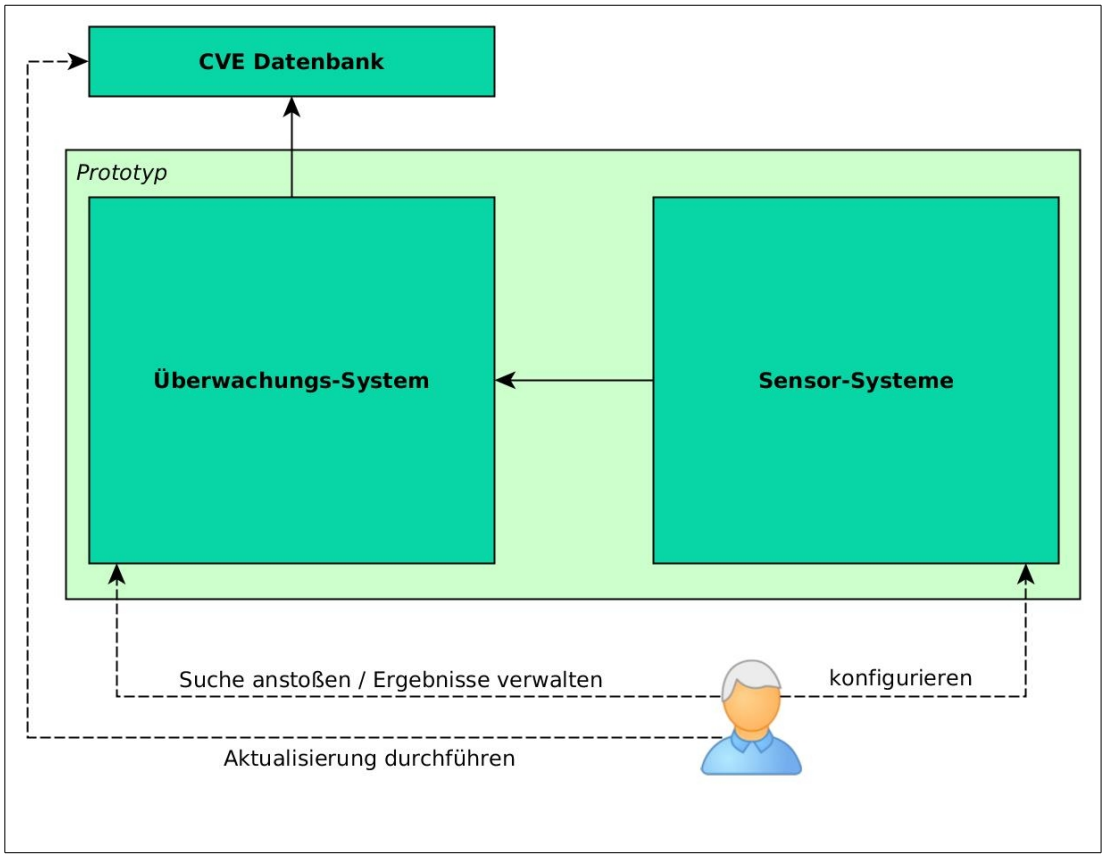

Abbildung 4.2: Kontextsicht

4.3 Verteilungssicht

In der Verteilungssicht werden die wesentlichen technischen Rahmenbedingungen dargestellt, in denen der Prototyp zur Ausführung kommt. Gegeben sei ein lokales Netzwerk mit den exemplarischen Subnetzwerken A, B und C, die überwacht werden sollen.
In jedem Subnetzwerk wird auf einem Host jeweils ein *Sensor-System* betrieben. Damit der Sensor seiner Aufgabe nachkommen kann und den

Netzwerkverkehr aller Hosts im Subnetzwerk aufzeichnen kann, ist es erforderlich, das Subnetzwerk dementsprechend zu konfigurieren.
Hierbei besteht die Möglichkeit, den Sensor entweder direkt im Datenstrom zu platzieren (Inline-Sensor) oder dem Sensor Kopien der Daten zur Verfügung zu stellen (Out-of-Line-Sensor), beispielweise über den Mirrorport eines Switches. *[vgl. Kappes, 2013, S.233]*
Die mitgeschnittenen und vorgefilterten Daten werden anschließend vom Sensor mittels eines, mit dem AMQP-Protokoll arbeitenden Message Brokers zum zentralen *Überwachungs-System* für die weitere Evaluierung gesendet.
Da die Umsetzung aller Systeme des Prototypen in der Programmiersprache *Python 3* erfolgt, wird auf allen Hosts die entsprechende Laufzeitumgebung benötigt.

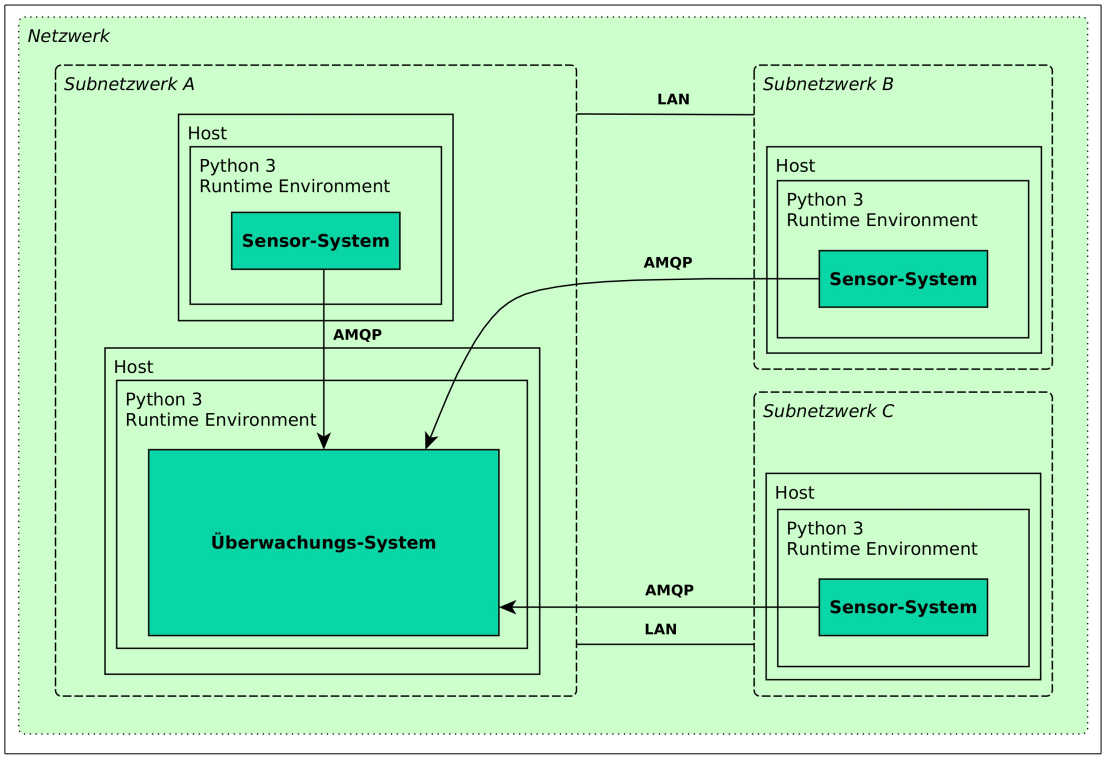

Abbildung 4.3: Verteilungssicht

4.4 Bausteinsicht

Die Bausteinsicht in *Abbildung 4.4* beschreibt den internen Aufbau des Prototypen und seine Architekturbausteine.

Das *Sensor-System* besteht im Wesentlichen nur aus einem Sensormodul, welches den Netzwerkverkehr analysiert, relevante Software-Update-Anfragen herausfiltert und an das zentrale *Überwachungs-System* übermittelt.

Das zentrale *Überwachungs-System* hingehen besteht aus mehrere Modulen:

Empfänger: Nachrichten der umliegenden *Sensor-Systeme* werden an den Empfänger gesendet. Diese Nachrichten enthalten zum Beispiel Informationen, welches Update, wann und von welchem Host angefragt wurde. Der Empfänger schreibt diese Information in die *Software-Datenbank*.

SW-Datenbank: In der Software-Datenbank werden die durch die *Sensor-Systeme* identifizierten Update-Anfragen persistiert.

Auswertungsmodul: Das Auswertungsmodul stellt die Kern-funktionen bereit, um die SW-Datenbank zu verwalten und die identifizierten Anwendungen mit Hilfe einer CVE-Datenbank auf bekannte Sicherheitslücken hin zu prüfen. Wird beispielsweise für einen Update-Eintrag in der Software-Datenbank eine Sicherheitslücke gefunden, wird die entsprechende CVE-ID durch das Auswertungsmodul zugeordnet bzw. vermerkt.

Konsolenanwendung: Die Konsolenanwendung ist die Benutzerschnittstelle für das Auswertungsmodul. Mit Hilfe einer überschaubaren Zahl von Parametern werden die bereits oben erwähnten Untersuchungen auf Sicherheitslücken angestoßen oder Einträge von Interesse angezeigt.

JSON-Schnittstelle: Diese Programmierschnittstelle bietet für die gesamte Funtionalität des Auswertungsmoduls Operationen an, die als Rückgabeformat JSON verwenden. Dadurch wird eine nachträgliche Anbindung einer grafischen Benutzerschnittstelle oder anderer Systeme vereinfacht.

CVE-Datenbank: Diese Datenbank enthält die gesammelten Informationen über bekannte Sicherheitslücken in Softwareprodukten und ist nicht Teil des Prototypen. Die eingesetzte CVE-Datenbank wird in *Kapitel 5* näher beschrieben.

Für den *Zugriff* auf den Prototypen, sei es die Konfiguration der Sensoren oder die Benutzung der Konsolenanwendung, ist die Verwendung einer SSH-Sitzung vorgesehen.

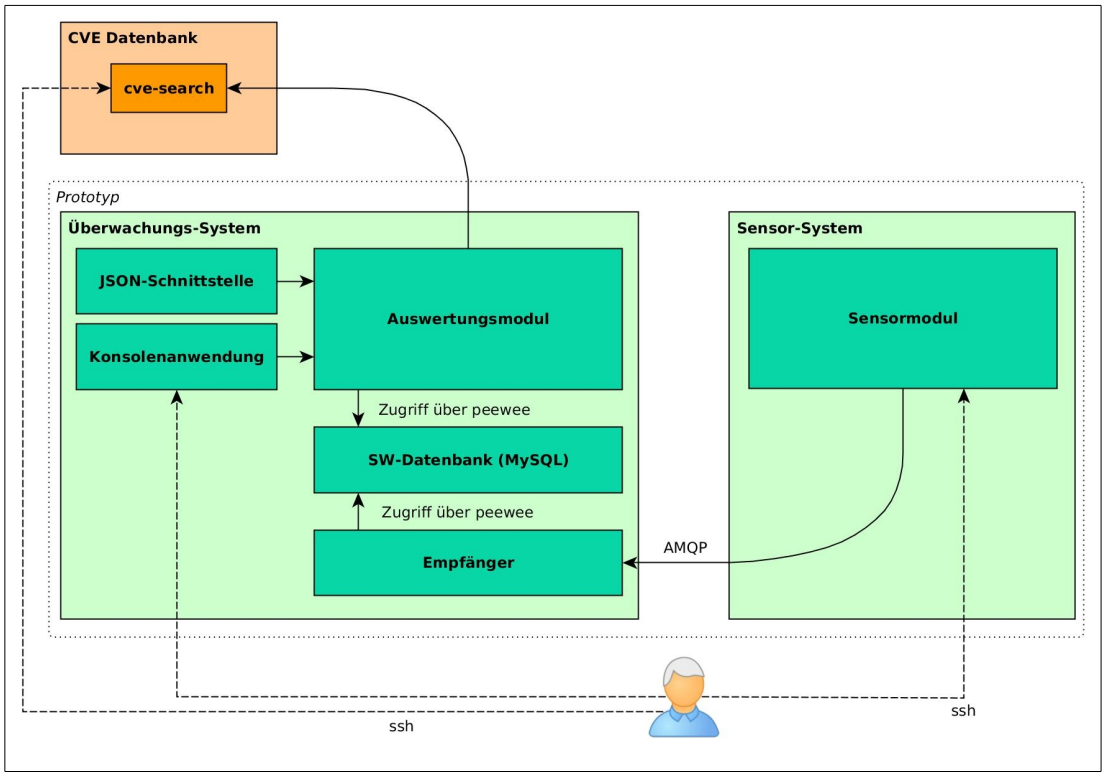

Abbildung 4.4: Bausteinsicht

4.5 Sequenzdiagramm

Das in *Abbildung 4.5* dargestellte Sequenzdiagramm zeigt die Interaktionen zwischen den Modulen des *Überwachungs-Systems* und *Sensor-Systems*, um den zeitlichen Ablauf von Aktionen zu verdeutlichen.

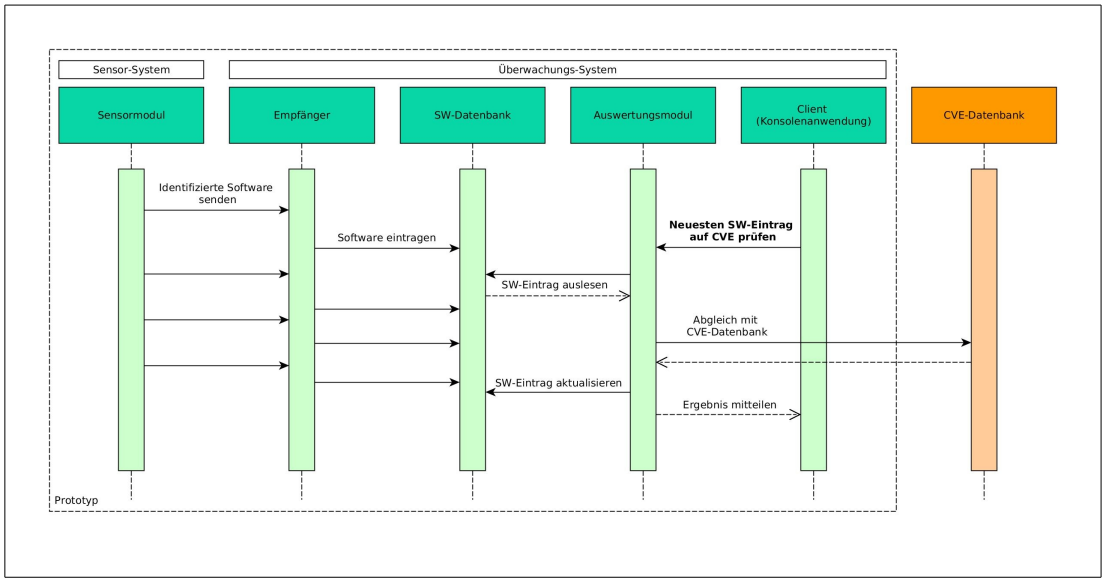

Abbildung 4.5: Sequenzdiagramm

Im Diagramm wird der beispielhafte Anwendungsfall betrachtet, in welchem der Benutzer über die Konsolenanwendung anfordert, den neuesten Software-Eintrag in der Software-Datenbank auf eine mögliche Sicherheitslücke hin zu überprüfen.

5. Implementierung

In diesem Kapitel soll die Implementierung des Prototypen betrachtet werden. Zunächst werden die gewählte Technologien und ihr Einsatzgebiet innerhalb des Prototypen vorgestellt *(Abschnitt 5.1 und 5.2)*. Die Beschreibung der Implementierung setzt am *Sensor-System* an *(Abschnitt 5.3)* und beginnt mit einer Analyse, woran die Update-kommunikation von Linux-Systemen zu erkennen ist. Daraufhin wird ein möglicher Algorithmus zur Filterung zunächst skizziert und anschließend die konkrete Umsetzung vorgestellt.
Da die durch die Filterung gewonnenen Daten an das zentrale *Überwachungs-System (Abschnitt 5.4)* gesendet werden sollen, wird betrachtet, wie die Übermittlung und die zentrale Datenhaltung gehandhabt werden. Zum Schluss sollen die Programmier- und Benutzer-Schnittstelle des *Überwachungs-Systems* sowie seine Anbindung an die externe CVE-Datenbank beschrieben werden.
Des Weiteren wird im Verlauf dieses Kapitels die Umsetzung von in *Kapitel 3* definierten funktionalen Anforderungen mit einem kurzen Vermerk gekennzeichnet und gegebenenfalls kurz begründet.

5.1 Programmiersprache

Die Implementierung des Prototypen erfolgt in *Python 3*. *Python* gilt als eine einfach erlernbare und weit verbreitete dynamisch typisierte Programmiersprache, die eine übersichtliche und strukturierte Syntax bietet. Sie unterstützt objektorientierte, aspektorientierte, strukturierte sowie funktionale Programmierung, sodass je nach Problemstellung ein passendes Paradigma gewählt werden kann. [vgl. Steyer, 2018, S.2 ff.] Darüber hinaus existiert eine Reihe von Bibliotheken und Frameworks, die sich gut zur Umsetzung des Prototypen eignen.

5.2 Bibliotheken und Frameworks

In diesem Abschnitt werden die wesentlichen Bibliotheken und Frameworks vorgestellt und ihr Anwendungsgebiet innerhalb des Prototypen beschrieben.

5.2.1 Datenformate

json - Die *JavaScript Object Notation* ist ein leichtgewichtiges Datenaustauschformat. Mit der JSON-Library von *Python* lassen sich auf einfache Weise Dictionaries in eine JSON-Repräsentation überführen, um diese im Anschluss zum Beispiel über ein Netzwerk zu übertragen. Im Prototypen wird das JSON-Format immer dann verwendet, wenn ein *Sensor-System* Informationen über ein identifiziertes Softwareupdate an das zentrale *Überwachungs-System* sendet.

5.2.2 Kommandozeilen-Interface

argparse – Dieses Modul vereinfacht die Programmierung von Kommandozeilen-Interfaces. Nach der Definition der gewünschten Argumente kümmert sich *argparse* um das richtige Parsen und generiert außerdem help- und usage-Nachrichten sowie Fehlermeldungen bei der Übergabe ungültiger Argumente.
argparse kommt im Prototypen vor allem in der Konsolenanwendung des *Überwachungs-Systems* zum Einsatz, wo dem Benutzer eine Vielzahl von Funktionen zur Verfügung gestellt werden.

tabulate – Hierbei handelt es sich um eine Library, mit der sich auf einfache Weise die Darstellung von Tabellen in Konsolen realisieren lässt. *Tabulate* kommt in der Konsolenanwendung des *Überwachungs-Systems* zum Einsatz, wenn Einträge aus der SW-Datenbank tabellarisch ausgegeben werden sollen.

5.2.3 Message Broker

RabbitMQ – Ein *Open-Source Message Broker* der unter anderem das *Advanced Message Queuing Protocol (AMQP)* implementiert. *RabbitMQ* wickelt im Prototypen den Nachrichtenaustausch zwischen Sensoren und dem *Überwachungs-System* ab (siehe *Abbildung 5.2.3*). Der *RabbitMQ-Server* kann beispielsweise auf dem Host des *Überwachungs-Systems* betrieben werden und empfängt Nachrichten von den *Sensor-Systemen*. Diese verwenden das Python-Modul *pika,* welches den *RabbitMQ-Client* implementiert. Das Empfänger-Modul des *Überwachungs-Systems* verwendet ebenfalls *pika* für das Empfangen der Nachrichten vom RabbitMQ-Server.

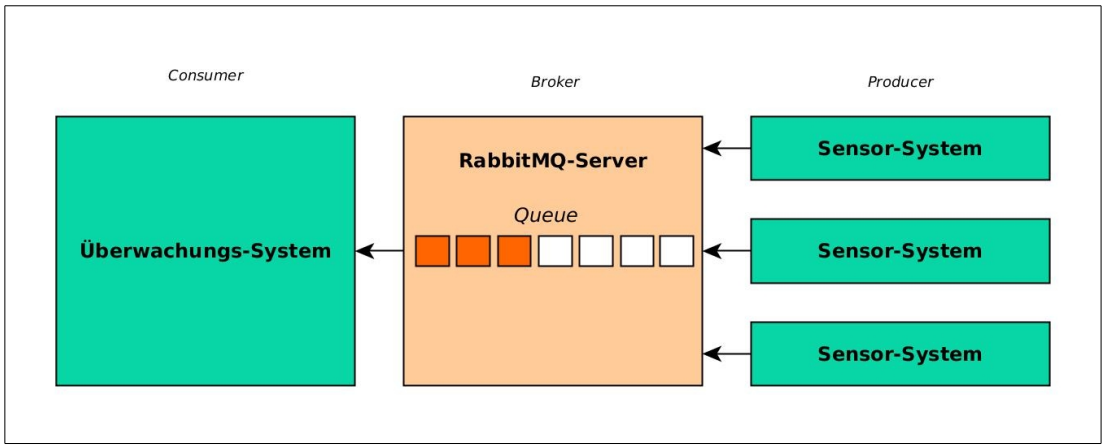

Abbildung 5.2.3: Nachrichten der Sensoren an das Überwachungs-System werden zunächst in der Queue des zentralen RabbitMQ-Servers eingereiht und dann an das Überwachungs-System zur weiteren Verarbeitung zugestellt.

5.2.4 Zentrale Datenhaltung

Peewee – Peewee ist eine leichtgewichtige Object Rational Mapping (ORM) Library und unterstützt SQLite, MySQL sowie Postgresql. Die Anbindung und Verwaltung der SW-Datenbank des *Überwachungs-Systems* wird über Peewee realisiert.

5.2.5 Sniffer

Scapy – Ein auf *Python* basierendes Programm und Library, mit der Netzwerkpakete verschiedenster Protokolle unter anderem empfangen, dekodiert und manipuliert werden können.
Das *Sensor-System* verwendet *Scapy* zum Mitschneiden des Netzwerkverkehrs und für eine erste grobe Filterung, bevor weitere Analysen eingeleitet werden.

5.2.6 Job Queue

RQ – *Redis Queue* ist eine Python-Library zur Realisierung von Job-Warteschlangen und Job-Abarbeitung durch sogenannte Worker im Hintergrund. Sie wird im *Sensor-System* verwendet, um das Senden von Nachrichten an das *Überwachungs-System* in Auftrag zu geben.

5.2.7 Lokale CVE-Datenbank

cve-search – Ein in *Python* geschriebenes Tool, um *CVE* und *CPE* in eine lokale *MongoDB*-Datenbank zu importieren und durchsuchbar zu machen. Durch Nutzung von *cve-search* werden Suchanfragen nicht nur schneller ausgeführt, sondern auch diskret behandelt, da nur eine lokale Datenbank durchsucht wird.
Das *Überwachungs-System* des Prototypen verwendet *cve-search*, um durch Sensoren erkannte Software auf bekannte Sicherheitslücken hin zu überprüfen.

5.3 Sensor-System

5.3.1 Analyse: Updates unter Linux

Um Anwendungen anhand von Update-Kommunikation zu identifizieren, soll zunächst betrachtet werden, was im Zuge eines Updates an Netzwerkverkehr anfällt. Die Herleitung eines möglichen Algorithmus soll anhand eines konkreten durchgeführten Updates veranschaulicht werden. Das in folgendem Beispiel verwendete System läuft mit der Linux-Distribution *Ubuntu* in der Version *16.04.3 LTS*. Für die Analyse sollen alle Systemkomponenten, für die ein Update zur Verfügung steht, geupdatet werden. Der erste Schritt, der unternommen werden muss, ist die Aktualisierung der Paketlisten, dies geschieht mit dem Befehl:

```
apt-get update
```

Abbildung 5.3.1.1: Befehl zur Aktualisierung der Paketquellen unter Ubuntu

Der Mitschnitt der Netzwerkverkehrs offenbart, dass durch Ausführung des Befehls einige HTTP-GET-Requests abgesetzt wurden – hier ein Request als Beispiel:

```
GET /ubuntu/dists/xenial-security/InRelease
HTTP/1.1\r\nHost: security.ubuntu.com\r\nCache-Control:
max-age=0\r\nAccept: text/*\r\nUser-Agent: Debian APT-
HTTP/1.3 (1.2.24)
```

Abbildung 5.3.1.2: Einer der abgesetzten HTTP-Requests durch die Aktualisierung der Paketquellen

Während die angefragte URI noch keinen Hinweis darauf liefert, was genau geupdated werden soll, fällt der verwendete User-Agent `Debian APT-HTTP/1.3 (1.2.24)` auf. Und tatsächlich eignet sich der User-Agent

als erstes Merkmal zur Filterung nach Updatekommunikation, wie im nächsten Schritt zu sehen sein wird.
Nachdem die Paketquellen aktualisiert worden sind, kann der eigentliche Updatevorgang angestoßen werden. Mit dem folgenden Befehl werden bereits installierte Pakete auf die aktuellste Version geupdatet:

```
apt-get upgrade
```

Abbildung 5.3.1.3: Befehl zur Aktualisierung aller installierten Pakete

Die Ausführung führt erneut zu einer Reihe von HTTP-GET-Requests, die dieses Mal aber aussagekräftiger sind, wie folgender Ausschnitt zeigt:

```
GET /ubuntu/pool/main/p/perl/perl-base_5.22.1-
9ubuntu0.2_amd64.deb HTTP/1.1\r\nHost:
archive.ubuntu.com\r\nUser-Agent: Debian APT-HTTP/1.3
(1.2.24)
```

Abbildung 5.3.1.4: HTTP-Request unter Ubuntu 16.04.3 LTS

Erneut wird der bereits zuvor beobachtete User-Agent verwendet. Außerdem kann nun aus der angefragten URI abgeleitet werden, für welche Komponente ein Update angefordert wurde, nämlich *Perl*. Die Version, auf welche aktualisiert wird, ist ebenfalls sichtbar: *5.22.1.*

Unter der Annahme, dass angefragte Updates auch vollständig runtergeladen und installiert werden, ist diese Art der Analyse eine Möglichkeit, Rückschlüsse auf installierte Software und ihre Versionsnummer zu ermöglichen.

Erwähnenswert ist ebenfalls, dass selbiger HTTP-GET-Request reproduziert werden kann, wenn Perl manuell nachinstalliert wird, zum Beispiel mittels des Befehls `apt-get install perl`. So werden mit der obigen Analyse also auch Neuinstallationen erkennbar.

Was nun untersucht werden soll, ist, ob die hier erkannten Muster auch auf andere Linux-Derivate und deren Distributionen übertragbar sind. Im Rahmen dieser Arbeit wurden für sechs gängige Linux-Derivate, stellvertretend jeweils eine Distribution, auf das oben beschriebene Updateverhalten hin untersucht. Diese Distributionen sind:

- Ubuntu 16.04.3 LTS *(Ubuntu-Derivate)*
- openSUSE Leap 42.3 *(SUSE-Derivate)*
- Fedora 26 *(Fedora-Derivate)*
- CentOS Linux 7 *(RHEL-Derivate)*
- Manjaro Linux *(Arch-Derivate)*
- Debian GNU/Linux 9 *(Debian-Derivate)*

An dieser Stelle soll zur Veranschaulichung das obige Beispiel mit dem Update von *Perl* erneut aufgenommen werden. Die gesendeten HTTP-GET-Requests der verschiedenen Distributionen sind in den *Abbildungen 5.3.1.5 bis 5.3.1.9* zu sehen.

```
GET /update/leap/42.3/oss/x86_64/perl-base-5.18.2-
9.1.x86_64.rpm HTTP/1.1\r\nUser-Agent: ZYpp 16.15.6 (curl
7.37.0) openSUSE-Leap-42.3-x86_64\r\nHost:
download.opensuse.org\r\nX-ZYpp-AnonymousId: eb8d6aa4-
88ae-4c95-97fe-68fe2c76306c\r\nX-ZYpp-DistributionFlavor:
mini\r\nAccept: */*, application/metalink+xml,
application/metalink4+xml
```

Abbildung 5.3.1.5: HTTP-Request unter openSUSE Leap 42.3 (SUSE-Derivate)

```
GET /linux/fedora/linux/updates/26/x86_64/Packages/p/
perl-5.24.3-396.fc26.x86_64.rpm HTTP/1.1\r\nHost:
ftp.uni-bayreuth.de\r\nUser-Agent: dnf/2.6.3\r\nAccept:
*/*
```

Abbildung 5.3.1.6: HTTP-Request unter Fedora 26 (Fedora-Derivate)

```
GET /centos/7.4.1708/os/x86_64/Packages/perl-5.16.3-
292.el7.x86_64.rpm HTTP/1.1\r\nUser-Agent:
urlgrabber/3.10 yum/3.4.3\r\nHost:
mirror.fra10.de.leaseweb.net\r\nAccept: */*
```

Abbildung 5.3.1.7: HTTP-Request unter CentOS Linux 7 (RHEL-Derivate)

```
GET /stable/core/x86_64/perl-5.26.1-2-x86_64.pkg.tar.xz
HTTP/1.1\r\nHost: repo.manjaro.org.uk\r\nUser-Agent:
pacman/5.0.2 (Linux x86_64) libalpm/10.0.2\r\nAccept: */*
```

Abbildung 5.3.1.8: HTTP-Request unter Manjaro Linux (Arch-Derivate)

```
GET /pool/updates/main/p/perl/perl_5.24.1-
3%2bdeb9u3_amd64.deb HTTP/1.1\r\nHost:
security.debian.org\r\nUser-Agent: Debian APT-HTTP/1.3
(1.4.8)
```

Abbildung 5.3.1.9: HTTP-Request unter Debian GNU/Linux 9 (Debian-Derivate)

Wieder ist zu erkennen, dass der Softwarename und die Versionnummer in der URI erkennbar sind und mit einem geeigneten Verfahren zu extrahieren wären. Ebenso sind weitere betriebssystemspezifische User-Agents hinzugekommen, die zur Filterung durch den Sensor genutzt werden können, um Updates bei unterschiedlichen Distributionen zu erkennen:

User-Agent	Distribution	Derivate
Debian APT-HTTP	Ubuntu 16.04.3 LTS	Ubuntu
Debian APT-HTTP	Debian GNU/Linux 9	Debian
ZYpp	openSUSE Leap 42.3	SUSE
dnf	Fedora 26	Fedora
urlgrabber	CentOS Linux 7	RHEL
pacman	Manjaro Linux	Arch

Abbildung 5.3.1.10: User-Agents verschiedener Distributionen und Derivate

Die hier beschriebenen Update-Muster sind, mit wenigen Ausnahmen, bei allen Distributionen beim Updaten verschiedenster Anwendungen beobachtbar. Eine Ausnahme bildet hier der weit verbreitete *Flash Player* von *Adobe*. In *Abbildung 5.3.1.11* ist zu sehen, dass anstelle einer Versionsnummer eine Datumsangabe zur Beschreibung der Version verwendet wird. Allerdings ist die Zuordnung von Datumsangaben zu Versionsnummer wegen fehlender Quellen zum Nachschlagen nicht realisierbar.

```
GET /pool/partner/a/adobe-flashplugin/adobe-
flashplugin_20180410.1.orig.tar.gz HTTP/1.1\r\nHost:
archive.canonical.com\r\nUser-Agent: Debian APT-HTTP/1.3
(1.2.24)
```

Abbildung 5.3.1.11: HTTP-Request beim Updaten des Adobe Flash Players

Der *Flash Player* wäre also mit dem hier vorgestellten Verfahren nicht überprüfbar. Es ist davon auszugehen, dass auch weitere Updates über URIs angefragt werden, die nicht der hier erwarteten Formatierung entsprechen. Allerdings waren weitere Ausnahmen im Rahmen der Analyse nicht zu beobachten.

Aus den oben beschriebenen Beobachtungen lassen sich nun die Kriterien für die Filterung von Update-Anfragen durch einen Scanner wie folgt zusammenfassen:

- Es wird nach *TCP*-Paketen auf dem Port gefiltert, auf dem ausgehende *HTTP*-Requests erwartet werden
- Es sind lediglich *HTTP-GET-Requests* von Interesse
- Filterung der *User-Agents* anhand einer vorgegebenen Whitelist
- *Analyse* der URI, ob tatsächlich eine Update-Anfrage vorliegt
- *Extrahierung* der Softwarebezeichnung und Versionsnummer

In *Abbildung 5.3.1.12* wurde aus diesen Kriterien ein Aktivitätsdiagramm abgeleitet, um zu veranschaulichen, wie die Struktur eines Algorithmus aussehen könnte, der Updates erkennt und benennt.
Die letzten beiden Punkte, die Untersuchung der URI und die Extrahierung von Softwarebezeichnung und Versionsnummer, wurden bisher nicht näher beleuchtet. Dies soll im nächsten Abschnitt erfolgen.

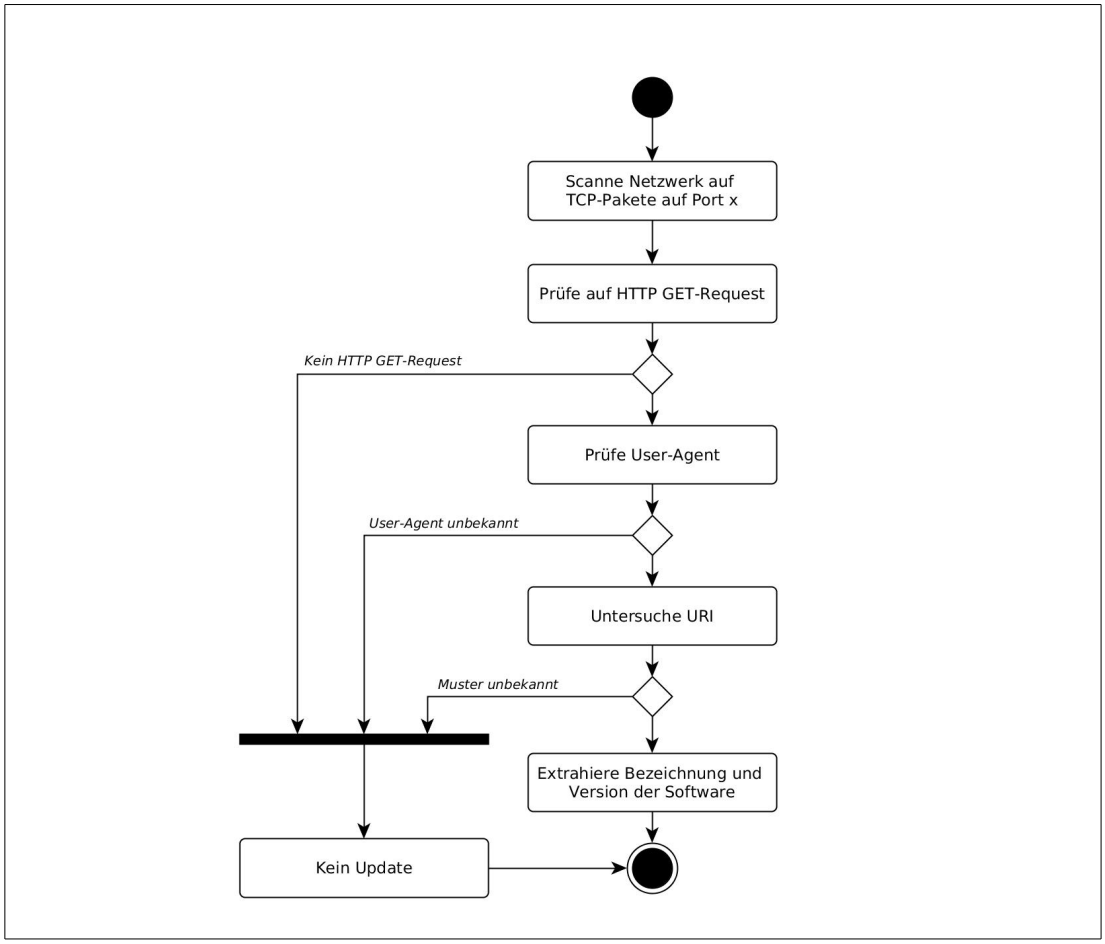

Abbildung 5.3.1.12: Aktivitätsdiagramm zur Skizzierung einer möglichen Filterung des Netzwerkverkehrs auf Updateanfragen

5.3.2 Algorithmus zur Auswertung der URI

Anhand einer beispielhaften URI soll Schritt für Schritt die Funktionsweise des Algorithmus erklärt werden. Ausgehend vom Aktivitätsdiagramm in *Abbildung 5.3.1.12* setzt der Algorithmus beim Punkt „Untersuche URI“ an und endet nach der Extraktion der Softwarebezeichnung und Versionsnummer, sofern die URI nicht vorher verworfen wird.
Die folgende Beschreibung stützt sich auf dem in *Abbildung 5.3.2* gezeigten Diagramm. Die beschriebenen Schritte finden sich in diesem wieder und werden dort möglichst nachvollziehbar visualisiert.

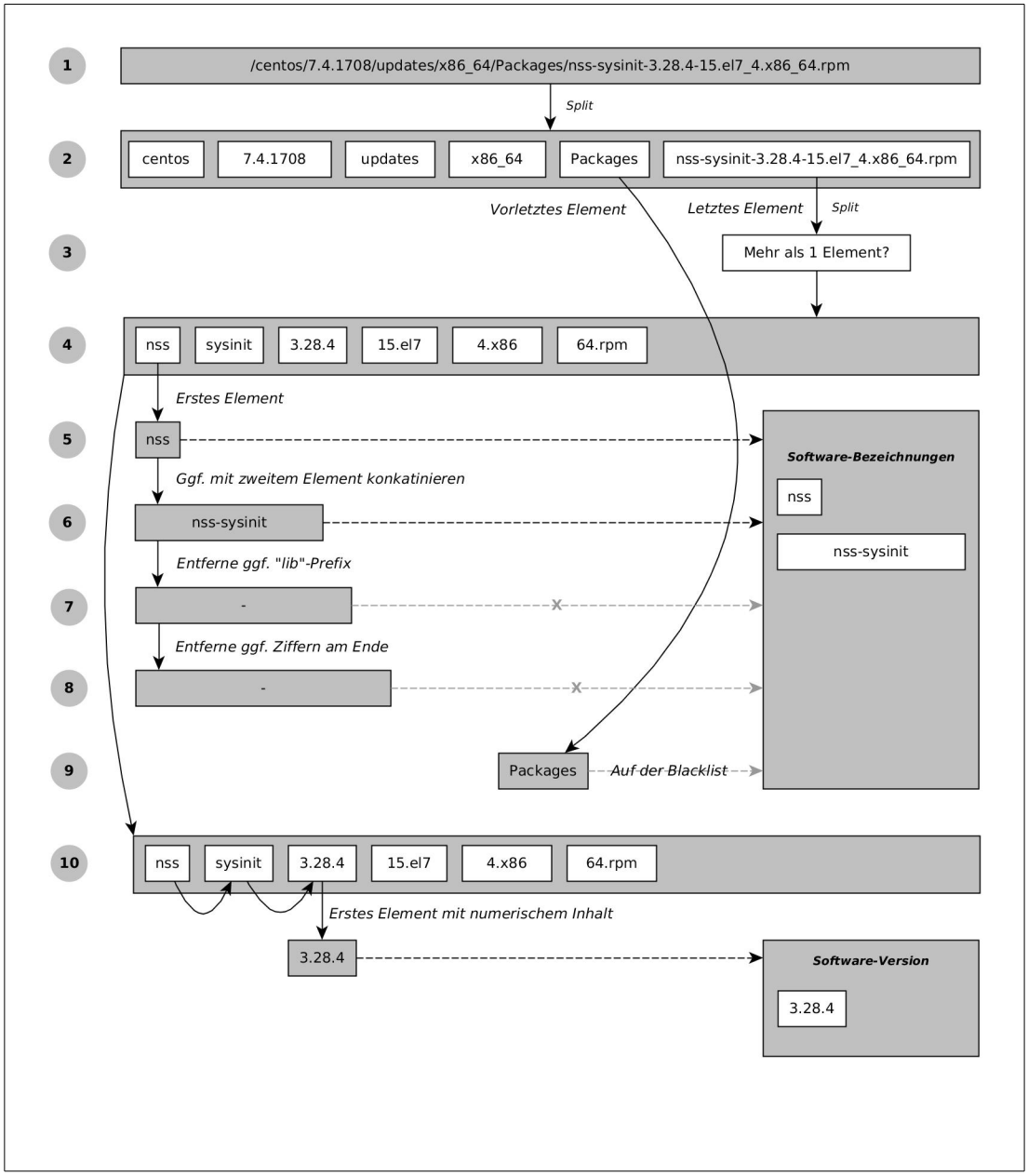

Abbildung 5.3.2: Schrittweise Auswertung einer beispielhaften URI durch den Algorithmus zur Extraktion der Softwarebezeichnung und Versionsnummer

1. Ausgangspunkt sei folgende URI, die als String vorliegt: *„/centos/7.4.1708/updates/x86_64/Packages/nss-sysinit-3.28.4-15.el7_4.x86_64.rpm“*

Im ersten Schritt wird die URI anhand des „/“-Zeichens gesplittet, um so ein Liste von String-Elementen zu erhalten, die jeweils einen Teilpfad der URI repräsentieren.

2. Aus dieser Liste wird nun das letzte Element, also *nss-sysinit-3.28.4-15.el7_4.x86_64.rpm,* genommen und erneut gesplittet, dieses Mal anhand der Zeichen „-“, „_“ oder „%“. Jene Zeichen werden meistens dazu verwendet, um Teile innerhalb eines Dateinamen zu separieren, wie zum Beispiel die Softwarebezeichnung und Versionsnummer.

3. Aus dem Split ergibt sich eine neue Liste, die nun untersucht wird:
   Hat die Liste nur ein Element, handelt es sich hierbei höchstwahrscheinlich um keine Update-Anfrage, sondern zum Beispiel um das Updaten von Paketquellen. In diesem Fall wird die URI verworfen und der Algorithmus endet. Als Beispiel für eine URI die verworfen werden würde, wird hier nochmals auf die *Abbildung 5.3.1.2* verwiesen.

4. Der Algorithmus arbeitet ohne internes Wissen über existierende Anwendungen. Er leitet die Softwarebezeichnung also komplett aus dem Dateinamen ab, der in der URI enthalten ist. Da die Dateinamen, die im Rahmen der Update-URIs beobachtbar sind, keiner strikten Namenskonvention unterliegen, ist eine eindeutige und korrekte Extraktion einer Softwarebezeichnung nicht immer möglich. Damit der Abgleich mit einer CVE-Datenbank später nicht daran scheitert, dass sich der Algorithmus auf eine (falsche) Bezeichnung festlegen musste, ist eine gewisse Flexibilität nötig.
   Deshalb hat der Algorithmus am Ende seiner Untersuchung die Möglichkeit, mehrere Bezeichnungen zurückzugeben. Dies erhöht zwar den Aufwand beim Abgleich mit einer CVE-Datenbank, da alle Optionen überprüft werden müssen, senkt aber auch das Risiko eine Sicherheitslücke aufgrund einer falschen Bezeichnung zu übersehen.

5. Aus der Liste (die aus dem Split in Schritt 2 entstanden ist) wird immer das erste Element zur Menge der möglichen Bezeichnungen hinzugefügt.

6. Um zusammengesetzte Bezeichnungen ebenfalls zu erfassen, werden gegebenenfalls das erste und zweite Element aneinander konkatiniert und auch zur Menge hinzugefügt. Dies geschieht immer dann, wenn das zweite Element mit mindestens einem Buchstaben beginnt. Ein Beispiel für eine zusammengesetzte Bezeichnung wäre zum Beispiel „*net-tools*“.

7. Es kommt vor, dass Anwendungsupdates ausschließlich mit einem „lib“-Präfix angefordert werden, der aber nicht Teil der eigentlichen Bezeichnung ist (z.B. „*libsystemd“)*. Dieser Präfix wird gegebenenfalls in diesem Schritt entfernt und die daraus resultierende Bezeichnung als Option zur Menge hinzugefügt.

8. Es ist ebenfalls zu beobachten, dass gelegentlich einstellige Versionsnummern direkt am Ende der eigentlichen Bezeichnung angehängt werden. Diese müssen entfernt werden, da in der Regel im CPE-Format eine strikte Trennung von Bezeichnung und Version verfolgt wird. Beispiele für einen solche Fall wären „*python3*“ oder „*sqlite3*“.

9. Möglich ist auch, dass die Auswertung eines ungeeigneten Dateinamens Bezeichnungen liefern, die bei der weiteren Auswertung in einer CVE-Datenbank ins Leere laufen. Deswegen ist es ratsam, den Namen des übergeordneten Verzeichnisses in der URI als Option hinzuzufügen. Diese Namen werden oftmals als Produkt innerhalb des CPE-Formats geführt.
   Ein Beispiel für einen ungeeigneten Datei- aber verwertbaren Verzeichnisnamen wäre folgende URI:
   /ubuntu/pool/main/l/linux-atm/libatm1_2.5.1-1.5_amd64.deb
   Die Aufnahme von nicht aussagekräftigen Verzeichnissen, wie „*Packages*“ *oder „x86_64*“, werden durch eine Blacklist verhindert.

10. Dieser Schritt extrahiert die Versionsnummer. Der Algorithmus wählt hierbei das erste Listenelement mit einer Versionsnummer in Punktnotation. Falls eine solche Versionsangabe nicht vorhanden sein sollte, wird das erste Listenelement mit einer ganzzahligen Versionsangabe gewählt. Zur Verdeutlichung folgen hier zwei Beispiele:
    - *libnl-genl-3-200_3.2.27-1ubuntu0.16.04.1_amd64.deb* ergibt die Versionsnummer *3.2.27*
    - *systemd_229-4ubuntu21_amd64.deb* ergibt die Versionsnummer *229*

5.3.3 Duplikate und Blacklisting

Mit dem bis hierhin beschriebenen Verfahren ist ein Sensor in der Lage, aus dem Netzwerkverkehr die relevanten HTTP-GET-Requests zu isolieren und Softwarebezeichnungen und Versionsnummern zu extrahieren *(Umsetzung von FA4)*. Bevor die Weiterleitung an das zentrale *Überwachungs-System* erfolgen kann, ist jedoch noch eine letzte Filterung notwendig. So kommt es vor, dass Updateanfragen für ein und dieselbe Anwendung bei verschiedenen Hosts angefragt werden. Zur Verdeutlichung ist in *Abbildung 5.3.3* eine Reihe von hintereinander abgesetzten HTTP-GET-Requests aufgeführt, die beim Updaten von *file-magic* beobachtet werden konnten.
Für alle diese Anfragen würde der Sensor die gleichen Bezeichnungen und Versionsnummer extrahieren (*file, file-magic / 5.22*). Eine Weiterleitung solcher Duplikate an das Überwachungsssystem würde keinen Mehrwert an Informationen bieten und die SW-Datenbank und Evaluation nur unnötig behindern. Daher ist die Realisierung einer temporären Blacklist sinnvoll, um kürzlich identifizierte Update-Requests nicht erneut weiterzuleiten. Ein Update wird in dieser jeweils durch die Softwarebezeichnung, die Versionsnummer sowie die MAC- und IP-Adresse des Hosts, von dem die Anfrage ausging, repräsentiert.
Bevor der Sensor also nun ein identifiziertes Update weiterleitet, erfolgt ein Abgleich mit der Blacklist. Ebenso wird jenes Update zur Blacklist hinzugefügt. Bei einem negativem Abgleich erfolgt sinngemäß die

Weiterleitung an das zentrale *Überwachungs-System*. Wird das Update bereits in der Blacklist geführt, wird eine Weiterleitung unterbunden.

```
(1) GET /update/leap/42.3/oss/x86_64/file-magic-5.22-
13.1.x86_64.rpm HTTP/1.1\r\nUser-Agent: ZYpp 16.15.6
(curl 7.37.0) openSUSE-Leap-42.3-x86_64\r\nHost:
download.opensuse.org\r\nX-ZYpp-AnonymousId: eb8d6aa4-
88ae-4c95-97fe-68fe2c76306c\r\nX-ZYpp-DistributionFlavor:
mini\r\nAccept: */*, application/metalink+xml,
application/metalink4+xml

(2) GET /pub/opensuse/update/leap/42.3/oss/x86_64/file-
magic-5.22-13.1.x86_64.rpm HTTP/1.1\r\nRange: bytes=0-
262143\r\nUser-Agent: ZYpp 16.15.6 (curl 7.37.0)
openSUSE-Leap-42.3-x86_64\r\nHost: ftp.gwdg.de\r\nAccept:
*/*

(...)

(8) GET /pub/linux/opensuse/update/leap/42.3/oss/x86_64/
file-magic-5.22-13.1.x86_64.rpm HTTP/1.1\r\nRange:
bytes=0-262143\r\nUser-Agent: ZYpp 16.15.6 (curl 7.37.0)
openSUSE-Leap-42.3-x86_64\r\nHost: ftp.rz.uni-
wuerzburg.de\r\nAccept: */*
```

Abbildung 5.3.3: Hintereinander abgesetzte Update-Anfragen bei verschiedenen Hosts für ein und die selbe Anwendung

5.3.4 Daten eines Updates

Bei jedem identifizierten Update übermittelt ein *Sensor* neben der Softwarebezeichnung und Versionsnummer weitere Daten an das *Überwachungs-System*. Diese Daten können dazu dienen, Updates einem Host innerhalb der überwachten Netzwerke eindeutig zuzuordnen, oder um die SW-Datenbank des *Überwachungs-Systems* zielgerichtet zu durchsuchen.

- *Netzwerkbezeichnung*: Bei der Konfiguration des Sensors kann eine Bezeichnung für das Netzwerk, in dem der Sensor eingesetzt wird, vergeben werden. Diese Netzwerkbezeichnung wird für alle identifizierten Updates durch diesen Sensor verwendet.

- *Source MAC, IP und Port*: MAC, IP und Port des Hosts von dem eine Update-Anfrage abgesetzt wurde.

- *Destination IP, Port*: IP und Port des Hosts, an den die Update-Anfrage abgesetzt wurde.

- *Softwarebezeichnung*: Bezeichnung der Software, die der Sensor aus der Update-Anfrage extrahiert hat.

- *Softwareversion*: Versionsnummer der Software, die der Sensor aus der Update-Anfrage extrahiert hat.

- *Timestamp*: Zeitpunkt, zu dem die Update-Anfrage durch den Sensor aufgezeichnet wurde.

5.3.5 Übermittlung an das Überwachungs-System

Vor der Übermittlung werden die oben beschriebenen Informationen in das JSON-Format überführt (siehe auch *Abbildung 5.3.5.1*) und ein Übermittlungsauftrag in eine *Redis Queue* eingereiht, die durch Worker asynchron abgearbeitet wird. Dies soll verhindern, dass der Sensor unnötig blockiert, falls es zu Netzwerkproblemen während des Sendens kommt. *Abbildung 5.3.5.2* greift das Sequenzdiagramm aus *Kapitel 4* erneut auf und beleuchtet es unter dem Aspekt des Nachrichtenaustausches zwischen dem *Sensor-* und *Überwachungs-System*.

```
{"sw_version": "3.5.2", "sport": "54653", "unixtime":
1524770863, "network": "default_network", "sw_name":
"python", "dport": "http", "dip": "91.189.88.149",
"smac": "02:42:ac:11:ac:02", "sip": "172.17.0.2"}
```

Abbildung 5.3.5.1: Beispiel für Update-Daten im JSON-Format

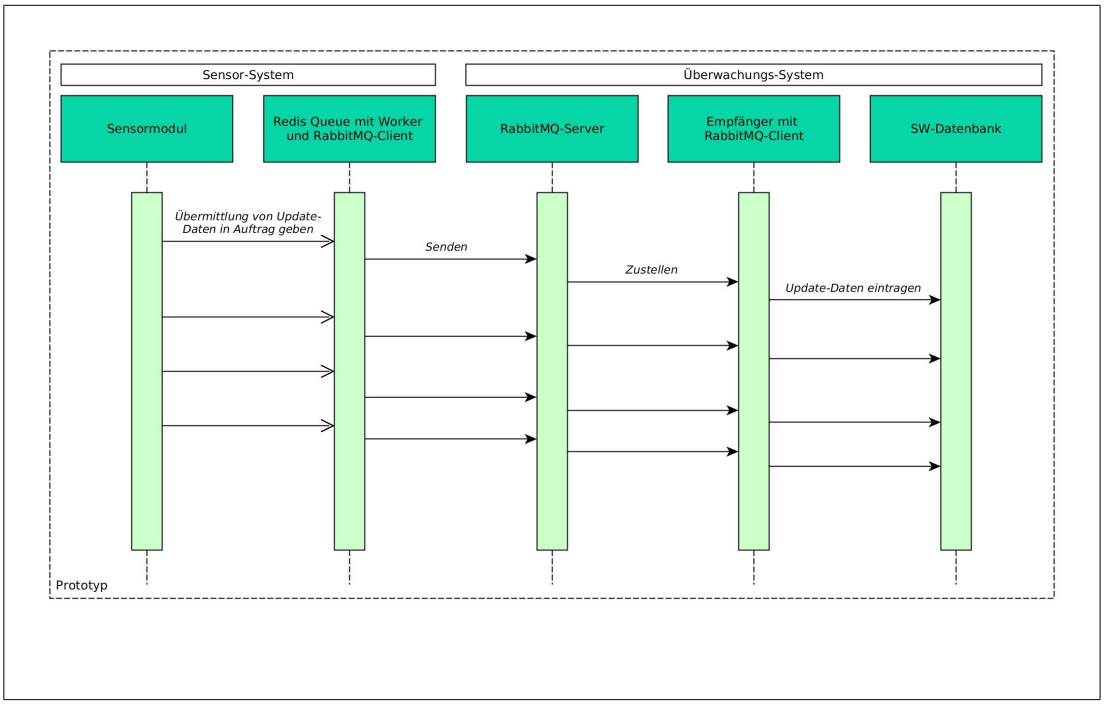

Abbildung 5.3.5.2: Asynchrone Abarbeitung von Übermittlungsaufträgen durch Redis Queue Worker und Zustellung über einen RabbitMQ-Server

5.3.6 Logging

Zur Überwachung der korrekten Funktionsweise des Sensors kann dieser im Verbose-Modus betrieben werden. In diesem Modus werden ausführliche Informationen in der Konsole ausgegeben, sodass der Nutzer nachvollziehen kann, welche Aktionen vom Sensor durchgeführt wurden *(Umsetzung von FA5)*.

Die Ausgabe setzt nach der Filterung auf bekannte User-Agents an (siehe auch *Abbildung 5.3.1.12*) und informiert, wie die mitgeschnittenen HTTP-GET-Requests ausgewertet wurden. Im Wesentlichen werden folgende Fragen zu jedem relevanten Request beantwortet:

- Um welchen HTTP-GET-Requests handelt es sich?
- Liegt ein Update vor?
- Wenn ja, welche Software und Version wurde ermittelt?
- Wie sieht der vollständige Datensatz zu diesem Request aus?
- Handelt es sich um ein Duplikat nach *Abschnitt 5.3.3*?

Folgende Abbildung zeigt eine beispielhafte Ausgabe des Verbose-Modus für einen mitgeschnittenen HTTP-GET-Request:

```
[09.10.2018 23:15:05][SNIFFED]
[Request] b'GET /ubuntu/pool/main/c/curl/curl_7.47.0-
1ubuntu2.9_amd64.deb HTTP/1.1\r\nHost: archive.ubuntu.com\r\nUser-
Agent: Debian APT-HTTP/1.3 (1.2.24)'
[Possible update identified]
[Software] {'curl'}
[Version] 7.47.0
[/SNIFFED]
[Generate JSON and submit] {"network": "default_network",
"sw_version": "7.47.0", "unixtime": 1539119705, "sip": "172.17.0.2",
"sw_name": "curl", "sport": "47882", "smac": "02:42:ac:11:00:02",
"dip": "91.189.88.162", "dport": "http"}
```

Abbildung 5.3.6.1: Beispielhafte Ausgabe des Sensors im Verbose-Modus

Das *Empfänger-Modul* des *Überwachungs-Systems* verfügt ebenfalls über einen Verbose-Modus, welcher Auskunft über die empfangenen Nachrichten der verteilten Sensoren gibt *(Umsetzung von FA6)*. Dieser kann dazu genutzt werden, um beispielsweise die korrekte Einstellung der Sensoren in der jeweiligen Konfigurationsdatei *(Umsetzung von FA3)* oder die Konfiguration der Sicherheitskomponenten im Netzwerk zu überprüfen.

```
[09.10.2018 23:15:06][RECEIVED]
{'sw_name': 'curl', 'smac': '02:42:ac:11:00:02', 'unixtime':
1539119705, 'sw_version': '7.47.0', 'dip': '91.189.88.162', 'dport':
'http', 'sport': '47882', 'network': 'default_network', 'sip':
'172.17.0.2'}
```

Abbildung 5.3.6.2: Im Verbose-Modus gibt das Empfänger-Modul des Überwachungs-Systems die Datensätze aus, die es von den Sensoren empfängt

5.4 Überwachungs-System

5.4.1 Zentrale Datenhaltung

Alle durch die *Sensor-Systeme* identifizierten Updates laufen im *Überwachungs-System* zusammen und werden vom *Empfänger-Modul* in einer MySQL-Datenbank persistiert. Zusätzlich zu den bereits in *Abschnitt 5.3.4* aufgeführten Update-Attributen werden lediglich zwei weitere Attribute für jeden Update-Eintrag in der Datenbank geführt:

- *Überprüft*: Sagt aus, ob für den Eintrag bereits ein Abgleich auf Sicherheitslücken mit der CVE-Datenbank stattgefunden hat.

- *CVE-ID*: Falls bei einem Abgleich mit der CVE-Datenbank für die gegebene Softwarebezeichnung und Versionsnummer bekannte Sicherheitslücken gefunden werden, werden diese hier aufgeführt.

Diese zusätzlichen Attribute sind notwendig, um die besonders sicherheitskritische Filterung nach bisher ungeprüften oder verwundbaren Systemen im Netzwerk zu ermöglichen *(siehe FA8)*.

5.4.2 CVE-Datenbank

Aus Sicherheitsgründen sollte der Abgleich von eingesetzter Software nicht mit einer im Internet lokalisierten CVE-Datenbank durchgeführt werden. Denn auch wenn solche Anfragen über eine verschlüsselte Verbindungen gesendet werden, könnten sie beispielsweise von der externen CVE-Datenbank geloggt werden. Geraten solche Informationen in die Hände potentieller Angreifer, könnten sie diese ausnutzen, um die Chancen für einen erfolgreichen Angriff zu erhöhen. Es sollte also im Sinne von Risikominimierung vermieden werden, dass jegliche Informationen über eingesetzte Software in einem Netzwerk nach außen kommuniziert werden. Aus diesem Grund wird eine lokale CVE-Datenbank aufgesetzt und an den Prototypen angebunden *(Umsetzung von FA7)*.
Das hierfür gewählte Tool *cve-search* bietet die notwendige Funktionalität, um anhand von CPE-Namen nach bekannten Sicherheitslücken für diese zu suchen. Da die Sensoren aus der Updatekommunikation nur die Softwarebezeichnung und Versionsnummer extrahieren können, ist eine Bildung von vollständigen CPE-Namen nicht möglich. Für die Suche stehen also nur die CPE-Attribute „*product*“ und „*version*“ zur Verfügung. Bei der Verwendung von *cve-search* ist aber auch die Angabe eines nur zum Teil vollständigen CPE-Namens möglich, wie in der folgenden Abbildung zu sehen ist:

```
./search.py -p python:3.5.2 -o cveid
```

Abbildung 5.4.2: Befehl um mit cve-search nach CVE-IDs für Python mit der Version 3.5.2 zu suchen und auszugeben

Leider bietet *cve-search* diese Funktionalität nicht in einer Programmierschnittstelle an. Um das Umschreiben und Ergänzen des cve-search-Quellcodes zu vermeiden, bringt das *Überwachungs-System* das *search.py*-Skript zur Ausführung und interpretiert seine Ausgabe. Dieser pragmatische, aber wenig elegante Lösungansatz sollte bei einem ernsthaften Einsatz des Prototypen überdacht und überarbeitet werden.

Die Aktualisierung der CVE-Datenbank wird, wie bereits in *Kapitel 4* erwähnt, nicht durch den Prototypen angestoßen. Der Updatevorgang, der je nach Größe der notwendigen Aktualisierung von wenigen Minuten bis zu mehreren Stunden dauern kann, muss vom Nutzer direkt über *cve-search* initiiert und überwacht werden.

5.4.3 Benutzerschnittstelle

Die Benutzerschnittstelle wird durch eine überschaubare Kommandozeilenanwendung realisiert *(Umsetzung von FA11)*. Die funktionalen Anforderungen verlangen im Wesentlichen die Ausgabe von identifizierter Software *(FA8)*, die Überprüfung von identifizierter Software auf Sicherheitslücken *(FA9)* und die Ausgabe von detaillierten Informationen zu einer Sicherheitslücke, im Falle einer verwundbaren Software auf einem System *(FA10).* Diese Anforderungen wurden entsprechend umgesetzt.

- **Ausgabe von identifizierter Software**

  Für die Ausgabe der von den Sensoren gesammelten Informationen verfügt die Kommandozeilenanwendung über drei hintereinander geschaltete Filterungsmöglichkeiten, die miteinander kombiniert werden können:

  1. Filterung anhand der im Sensor festgelegten Netzwerkbezeichnung; MAC-Adresse des Linux-Systems; keine Einschränkungen

  2. Zeitraum, in der Software identifiziert wurde; Anzahl der neuesten Einträge; ein bestimmtes Intervall von Einträgen

  3. Einträge, die noch nicht mit der CVE-Datenbank abgeglichen wurden; Einträge, die bereits überprüft wurden; Einträge, bei denen eine Sicherheitslücke diagnostiziert wurde

Folgende Abbildung zeigt ein Beispiel, wie ein Befehl zur Suche nach erkannten Sicherheitslücken auf Systemen im Netzwerk mit der Bezeichnung „*default_network*“ für die Einträge 80 bis 120 aufgebaut und das zurückgegebene Resultat aufbereitet ist.

```
./m_terminal.py -s network=default_network -r 80 120 -cveo
[Printing 2 entries]
  ID  NETWORK          MAC               IP          DATE                SOFTWARE    VERSION    CHECKED    CVE:CVSS
----  ---------------  ----------------  ----------  ------------------  ----------  ---------  ---------  ------------------
  81  default_network  02:42:ac:11:00:02  172.17.0.2  02.09.2018 13:23:55  libidn      1.32       True       CVE-2016-6263:5.0
                                                                                                           CVE-2016-6261:5.0
                                                                                                           CVE-2015-8948:5.0
                                                                                                           CVE-2016-6262:5.0
 104  default_network  02:42:ac:11:00:02  172.17.0.2  02.09.2018 13:23:56  libxml2     2.9.3      True       CVE-2016-3627:5.0
                                                                                                           CVE-2016-4449:5.8
                                                                                                           CVE-2016-4448:10.0
                                                                                                           CVE-2016-4447:5.0
                                                                                                           CVE-2016-3705:5.0
```

Abbildung 5.4.3.1: Tabellarisch aufbereitetes Resultat für die Sucheanfrage nach Einträgen aus dem Netzwerk mit der Bezeichnug „default_network“, im ID-Intervall 80 bis 120, für die Sicherheitslücken erkannt wurden

Durch die einfache tabellarische Ausgabe kann sofort zugeordnet werden, auf welchem System und in welchem Netzwerk die Sicherheitslücke vorhanden ist, sodass entsprechende Maßnahmen ergriffen werden können.

- **Abgleich von identifizierter Software mit der CVE-Datenbank**

  Die Überprüfung auf Sicherheitslücken für Software-Einträge geschieht nicht automatisch, sondern muss vom Nutzer initiiert werden. Zur Eingrenzung der zu überprüfenden Einträge kann auf die gleichen Filter wie zuvor bei der *Ausgabe von identifizierter Software* zurückgegriffen werden.
  Sobald der Abgleich gestartet wird, informiert die Kommandozeilen-anwendung in Echtzeit über den Fortschritt des Gesamtprozesses und über das Resultat jedes einzelnen Abgleichs.

```
./m_terminal.py -c all -r 28 29 -uco

PROGRESS      ID  NETWORK          IP          MAC                DATE                 SOFTWARE      VERSION  MESSAGE
----------  ----  ---------------  ----------  -----------------  -------------------  ----------  ---------  --------------
1/2           28  default_network  172.17.0.2  02:42:ac:11:00:03  01.07.2018 17:34:37  udev             229  [No CVE found]


PROGRESS      ID  NETWORK          IP          MAC                DATE                 SOFTWARE      VERSION  MESSAGE
----------  ----  ---------------  ----------  -----------------  -------------------  ----------  ---------  --------------
2/2           29  default_network  172.17.0.2  02:42:ac:11:00:02  01.07.2018 17:34:37  glibc            2.23  [CVE found &
Entry updated] CVE found: CVE-2016-3075: 5.0, CVE-2016-5417: 5.0
```

Abbildung 5.4.3.2: Resultat für den Abgleich der noch ungeprüften Einträge aus dem Intervall 28 bis 29

- **Ausgabe von Informationen zu Sicherheitslücken**
  Für jeden beliebigen Eintrag mit einer identifizierten Sicherheitslücke ist es unter Angabe der entsprechenden Eintrags-ID möglich, sich die entsprechenden CVE-Informationen ausgeben zu lassen. Neben den bereits in *Abschnitt 2.4* eingeführten Attributen werden auch die betroffenen Konfigurationen im CPE-Standard aufgeführt. In folgendem Beispiel sollen zu dem Eintrag mit der ID *„1617“*, für den eine bekannte Sicherheitslücke vorliegt, weitere Informationen eingeholt werden:

```
./m_terminal.py -id 1617
CVE    : CVE-2016-6252
DATE   : 2017-02-17 12:59:00.937000
CVSS   : 4.6
Integer overflow in shadow 4.2.1 allows local users to gain privileges
via crafted input to newuidmap.

References:
-----------
http://www.openwall.com/lists/oss-security/2016/07/19/6
http://www.openwall.com/lists/oss-security/2016/07/19/7
http://www.openwall.com/lists/oss-security/2016/07/20/2
http://www.openwall.com/lists/oss-security/2016/07/25/7
http://www.securityfocus.com/bid/92055
https://bugzilla.suse.com/show_bug.cgi?id=979282
https://github.com/shadow-maint/shadow/issues/27
https://security.gentoo.org/glsa/201706-02

Vulnerable Configs:
-------------------
cpe:2.3:a:shadow_project:shadow:4.2.1
```

Abbildung 5.4.3.3: Beispiel zur Ausgabe von Informationen zu der vorliegenden Sicherheitslücke für den Eintrag mit der ID „1617“

```
./m_terminal.py -h
usage: m_terminal.py
                    [-h] [-s SHOW] [-c CHECK] [-id CVEBYID] [-st]
                    [-r RANGE RANGE] [-dr DATERANGE DATERANGE] [-l LATEST]
                    [-co] [-uco] [-cveo] [-d]

optional arguments:

-h, --help
                        show this help message and exit

-s SHOW, --show SHOW
                        show database entries - valid arguments: all,
                        mac=1a:2b:3c:4d:5e:6f, network=abcdefghi

-c CHECK, --check CHECK
                        check entries in database for cves – valid
                        arguments: all, mac=1a:2b:3c:4d:5e:6f,
                        network=abcdefghi

-id CVEBYID, --cvebyid CVEBYID
                        enter update entry id to get detailed information
                        about all cve regarding this entry

-st, --stats            show database stats

-r RANGE RANGE, --range RANGE RANGE
                        range of database entries

-dr DATERANGE DATERANGE, --daterange DATERANGE DATERANGE
                        date range of database entries;
                        dateformat: dd-mm-yyyy

-l LATEST, --latest LATEST
                        n newest database entries

-co, --checkedonly      consider checked entries only

-uco, --uncheckedonly   consider unchecked entries only

-cveo, --cveonly        consider checked entries with identified cve only

-d, --details           display all attributes when showing database entries
```

Abbildung 5.4.3.4: Überblick über die Funktionen der Kommandozeilenanwendung

Zusammengefasst ist die Benutzerschnittstelle mit ihren drei Kernfunktionen,

- *Ausgabe von identifizierter Software*
- *Abgleich von identifizierter Software mit der CVE-Datenbank*
- *Ausgabe von Informationen zu Sicherheitslücken*

und den vorgestellten Filterungsmöglichkeiten als kompakt zu bezeichnen (siehe auch *Abbildung 5.4.3.4*). Es wird daher davon ausgegangen, dass ein Nutzer bereits nach einer kurzen Eingewöhnungsphase das volle Funktionspotential ausnutzen kann.

5.4.4 Programmierschnittstelle

Die Erweiterbarkeit des *Überwachungs-Systems* im Sinne von *NFA2* wird über die Programmierschnittstelle gewährleistet. Für alle angebotenen Operationen werden Vor- und Nachbedingungen definiert und teilweise auch geprüft. Ebenso wie bei der Benutzerschnittstelle können auch über die Programmierschnittstelle Update-Einträge nach definierten Filterkriterien durchsucht oder der Abgleich mit der CVE-Datenbank angestoßen werden *(Umsetzung von FA11)*. Die Ergebnisdatensätze werden hierbei im JSON-Format zurückgegeben, um eine Weiterverarbeitung zu vereinfachen.

6. Evaluation

Der in dieser Arbeit entwickelte Prototyp zur Identifizierung von verwundbarer Software auf Linux-Systemen anhand der Updatekommunikation soll nun in einem einfachen Testszenario erprobt und die dabei gewonnenen Daten anhand verschiedener Aspekte betrachtet werden. Dies soll einen Überblick über die Eigenschaften des Prototypen geben und eine grobe Einschätzung ermöglichen, wie praktikabel die hier entwickelte Analyse der Updatekommunikation tatsächlich ist. Hierfür sollen im Rahmen der Evaluation die folgenden Aspekte untersucht werden:

- **Quantität der Daten**
  Wie hoch ist die Anzahl der Datensätze, die ein Sensor bei einem durchgeführten Softwareupdate auf einem überwachten Host aufzeichnet? Wird jede Updatekommunikation erfasst?

- **Aufwand zur Auswertung der Daten**
  Wie hoch ist der Aufwand, um aus den Datensätzen Schlussfolgerungen auf verwundbare Systeme ziehen zu können?

- **Aussagekraft der Daten**
  Inwiefern sind die gewonnen Daten aussagekräftig? Werden auch inkorrekte Aussagen gemacht?

6.1 Testszenario

Für die Erhebung der Daten wird ein Testszenario gewählt, welches aus drei Schritten besteht. Zunächst soll auf einem durch einen Sensor überwachten Linux-System der Browser *Firefox* in einer veralteten Version installiert werden. Wie in *Abschnitt 5.3.1* festgestellt, unterscheiden sich Neuinstallationen von Updates hinsichtlich der Netzwerkkommunikation nicht, sodass bereits die Installation von *Firefox* vom Sensor erfasst werden sollte.
Im zweiten Schritt soll *Firefox* auf die aktuelle Version geupdatet werden und die Updatekommunikation erneut vom Sensor überwacht werden.
Im dritten Schritt werden die gesammelten Update-Einträge vom *Überwachungs-System* hinsichtlich vorhandener Sicherheitslücken überprüft.

6.2 Testumgebung

Zur Erhebung der Daten wird eine einfache Testumgebung eingerichtet. Dafür werden alle benötigten Komponenten auf einem Linux-Host (Hardware: i3-4005u CPU mit 4GB Arbeitsspeicher) installiert und konfiguriert:

- Ein zu überwachendes Linux-System wird innerhalb eines Docker-Containers betrieben. Der gesamte Netzwerkverkehr dieses Linux-Systems wird über die Bridge `docker0` des Hosts geleitet. Außerdem hat das Linux-System über den Host Zugang zum Internet.

- Ein Sensor wird auf dem Host betrieben und derart konfiguriert, dass er den Netzwerkverkehr der über `docker0` läuft, snifft. Wenn also beispielsweise Updatekommunikation stattfindet, kann der Sensor diese mitschneiden und auswerten.

- Auf dem Host werden ebenfalls das *Überwachungs-System* und die lokale *CVE-Datenbank cve-search* installiert.

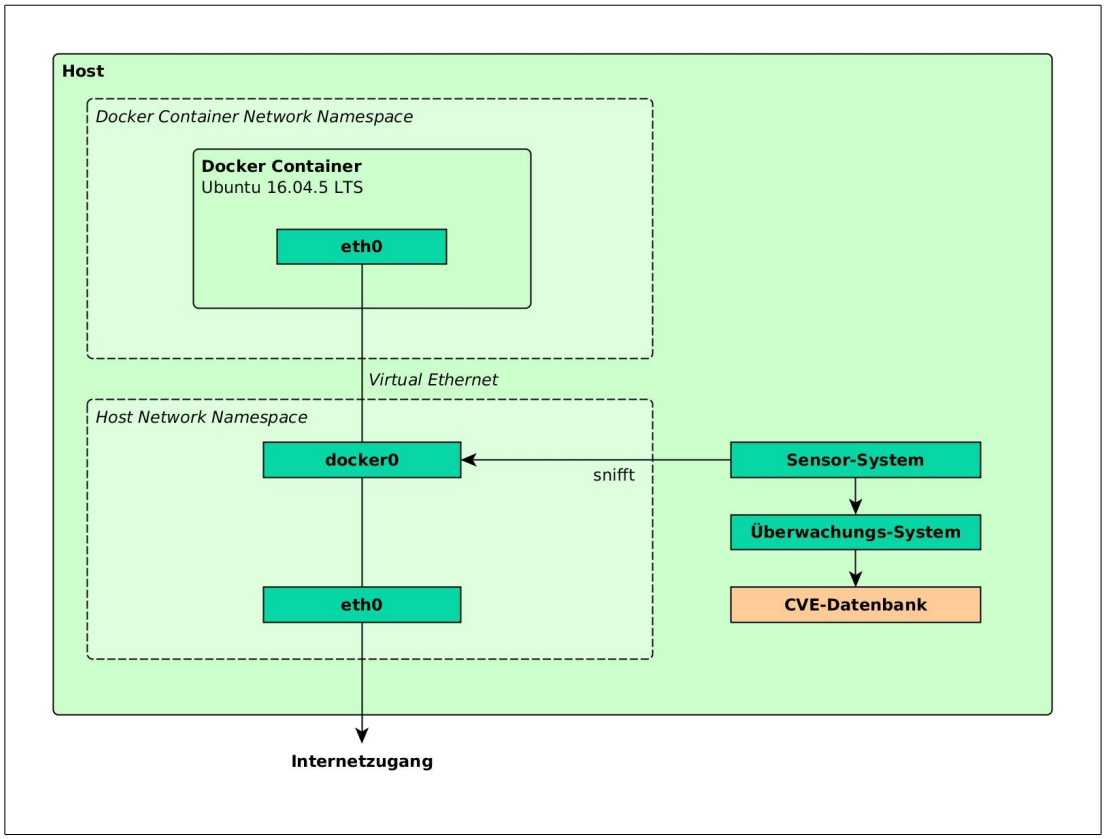

Abbildung 6.2: Aufbau der Testumgebung für die Evaluation

6.3 Quantität der Daten

Alle Daten, die durch die Sensoren gesammelt werden, müssen letztendlich auch ausgewertet werden, um aus ihnen Nutzen ziehen zu können. Daher soll untersucht werden, welche Menge an Daten in dem zuvor beschriebenen Szenario anfallen und ob auch jede Updatekommunikation aufgezeichnet wird. Als Referenz für die Überprüfung der Vollständigkeit der aufgezeichneten Updatekommunikation wird der Netzwerkverkehr über `docker0` mit Hilfe des Sniffing Tools *Wireshark* parallel mitgeschnitten und separat ausgewertet.

6.3.1 Vollständigkeit der Daten

Während der Installation von *Firefox* in der Version 45.0.2 erscheint in der Konsole die Information, dass bei der Installation 112 neue Pakete installiert werden (siehe *Abbildung 6.3.1.1*). Der Sniff von *Wireshark* zeigt, dass sogar 120 relevante Anfragen bei der Installation abgesetzt wurden. Dieser Wert sollte auch vom Sensor erreicht werden, um eine vollständige Erfassung aller Update-Requests nachzuweisen. Tatsächlich erreicht der Sensor diesen Wert nicht annähernd. So werden von diesem lediglich 71 Update-Requests erfasst (59,2%).

```
root@21ffc7724edf:/# apt-get install firefox=45.0.2+build1-0ubuntu1
(…)
The following NEW packages will be installed:
  dbus dh-python distro-info-data file firefox fontconfig
  (…)
  ucf x11-common xdg-user-dirs xml-core xul-ext-ubufox
0 upgraded, 112 newly installed, 0 to remove and 0 not upgraded.
```

Abbildung 6.3.1.1: Bei der Installation von Firefox 45.0.2 wird über die Installation von 112 neuen Pakete informiert.

Ähnlich verhält es sich bei dem darauf folgenden Update von *Firefox*.
In der Konsole erscheint die Information, dass insgesamt 90 Pakete neu installiert und eines geupgraded wird. Deckungsgleich mit diesen Angaben schneidet Wireshark 91 relevante Anfragen mit und der Sensor zeichnet dieses Mal 67 Update-Requests auf (73,6%).

```
root@21ffc7724edf:/# apt-get upgrade firefox
(…)
The following NEW packages will be installed:
  acl adwaita-icon-theme at-spi2-core colord colord-data
  (…)
  libxshmfence1 libxtst6 policykit-1 ubuntu-mono xkb-data
The following packages will be upgraded:
  firefox
1 upgraded, 90 newly installed, 0 to remove and 0 not upgraded.
```

Abbildung 6.3.1.2: Beim Update von Firefox auf die neueste Version, wird über die Installation von 91 Paketen informiert.

Angestoßen durch dieses Ergebnis wurde versucht die Ursache für die nicht identifizierten Anfragen zu lokalisieren. Dabei wurde das vom Sensor verwendete Sniffing-Tool *Scapy* auf das Filtern aller ausgehenden HTTP-GET-Requests konfiguriert und isoliert zur Ausführung gebracht. Das Firefox-Szenario wurde drei Mal durchgespielt und die aufgezeichneten Anfragen ausgewertet. Und auch hier waren die Ergebnisse ähnlich und stark schwankend. Bei der Installation von *Firefox* wurden im Mittel 77,8% *(Median 80,2%)* der Anfragen aufgezeichnet und beim Update 86,8% *(Median 71,7%)*. Besonders bei einem erhöhten Datendurchsatz schien *Scapy* nicht in der Lage zu sein, alle Netzwerkpakete filtern zu können. Es ist denkbar, dass es sich hierbei um einen bereits bekannten Bug handelt: „May miss packets under heavy load“ [SCAPY, 2018, BUGS].

Zusammengefasst kann also im Bezug auf die *Vollständigkeit der Daten* eine signifikante Diskrepanz in der Anzahl der durch das Linux-System abgesetzten und den durch den Prototypen erfassten Update-Anfragen ausgemacht werden. Durch das unzuverlässige Sniffen kann zu diesem Zeitpunkt der Netzwerkverkehr nur eingeschränkt ausgewertet werden, was eine lückenlose Erkennung von verwundbaren Systemen verhindert. Für einen ernsthaften Einsatz des Prototypen sollte also zunächst ein zuverlässiges Sniffen gewährleistet werden, um jeglichen Verlust von Informationen zu vermeiden.

6.3.2 Anzahl der Update-Einträge

Wie bereits in *Abschnitt 5.3.2* beschrieben werden aus den Softwarebezeichnungen, die aus den aufgezeichneten Update-URIs extrahiert werden, mehrere Varianten generiert. Dies soll die Chancen erhöhen, dass ein Abgleich mit der CVE-Datenbank auch dann gelingt, wenn die Softwarebezeichnungen in der Update-URI und im entsprechenden CPE-Namen unterschiedlich sind.
Bei der Installation und dem Update von Firefox wurden vom Sensor insgesamt 138 Update-Anfragen identifiziert. Dabei wurden 362 Softwarebezeichnungen generiert und entsprechende Update-Einträge im *Überwachungs-System* angelegt. Für jede identifizierte Update-Anfrage werden in diesem Szenario also im Schnitt etwa 2,6 Update-Einträge angelegt.

Es sollte also festgehalten werden, dass durch die Herangehensweise des Generierens von alternativen CPE-Namen damit gerechnet werden muss, dass stets mehrere Update-Einträge je Update-Anfrage angelegt werden. Die sich daraus ergebende Konsequenz ist ein erhöhter Aufwand in der Auswertung, da für eine aufgezeichnete Update-Anfrage alle daraus generierten Update-Einträge mit der CVE-Datenbank abgeglichen werden müssen. Allerdings ist das Generieren von alternativen CPE-Namen unumgänglich, wenn das in dieser Arbeit entwickelte Verfahren zur Extraktion von Softwarebezeichnungen und Versionsnummern angewendet wird.

Alternativ könnten folgende Ansätze in Betracht gezogen werden, um die Anzahl der zu prüfenden Update-Einträge zu reduzieren:

- Anlegen und Pflegen einer Datenbank, mit deren Hilfe von *Dateinamen* in den Update-URIs direkt auf den korrekten *CPE-Namen* abgebildet werden kann. Dadurch entfällt das Generieren von alternativen Softwarebezeichnungen und somit werden unnötige Update-Einträge vermieden.

- Anlegen einer *Whitelist,* die festlegt, für welche Software überhaupt Update-Einträge geführt werden sollen. Anstatt jedes noch so kleine Softwarepaket überwachen zu müssen, kann es gegebenenfalls sinnvoller sein, eine festgelegte Menge von Anwendungen zu überwachen, wie beispielsweise exponierte und kritische Dienste oder durch Mitarbeiter tagtäglich verwendete Anwendungen.

6.4 Aufwand zur Auswertung der Daten

Die Auswertung von Daten verbraucht immer Ressourcen. So ist es auch beim *Überwachungs-System*, wenn Update-Einträge mit der CVE-Datenbank abgeglichen werden müssen. Die Suche nach Sicherheitslücken für die vom *Überwachungs-System* übergebenen CPE-Namen benötigt Zeit, vor allem wenn große Mengen an Update-Einträgen zur Überprüfung vorliegen.

Auf dem für die Evaluation verwendeten Rechner werden für den Abgleich der 362 vorliegenden Update-Einträge mit der lokalen CVE-Datenbank *cve-search* 659 Sekunden benötigt. Für die Überprüfung eines Eintrags werden also etwa 1,8 Sekunden benötigt. Es wird allerdings nur noch ein Bruchteil dieser Zeit benötigt, wenn ein bereits überprüfter CPE-Name erneut geprüft werden soll, da die Ergebnisse während der Auswertung gecached werden. Dieser Fall kann beispielsweise dann eintreten, wenn mehrere überwachte Systeme ähnlich konfiguriert sind.
Angenommen es soll in einem Firmennetzwerk mit 100 weitestgehend identisch eingerichteten Hosts, für die jeweils 3000 Update-Einträge im *Überwachungs-System* hinterlegt sind, eine Überprüfung auf neue Sicherheitslücken durchgeführt werden. Dann würde der hier verwendete Rechner für die Überprüfung des ersten Hosts 90 Minuten benötigten und die verbleibenden 99 Hosts mit Hilfe der gecachten Ergebnisse effizienter abarbeiten. Wenn hingegen angenommen wird, dass diese 100 Hosts völlig unterschiedlich eingerichtet sind, würde die vollständige Überprüfung allerdings 150 Stunden in Anspruch nehmen.

Eine Möglichkeit den Auswertungsprozess zu beschleunigen, wäre die Auswertung durch mehrere *Threads* durchführen zu lassen. Die Implementation des *Überwachungs-Systems* verwendet momentan keine Threads, sodass die zur Verfügung stehenden Prozessorkerne nicht ausgelastet werden. Allerdings können die zu überprüfenden Einträge aufgeteilt und die Auswertung in mehreren Konsolen gleichzeitig angestoßen werden. Bei vier simultanen Auswertungen sind die Prozessor-Kerne des Testrechners voll ausgelastet und die 362 Einträge können in 290 Sekunden überprüft werden (etwa 0,8 Sekunden pro Eintrag).

In Bezug auf den *Aufwand zur Auswertung der Daten* lässt sich festhalten, dass eine Überwachung einer größeren Anzahl von Systemen nur dann effizient lösbar ist, wenn diese eine ähnliche Konfiguration aufweisen. Im Fall von sich stark unterscheidenden Konfigurationen können zwischengespeicherte Ergebnisse für bereits geprüfte CPE-Namen seltener wiederverwendet werden und es müssen häufiger Abfragen an die CVE-Datenbank gestellt werden.

6.5 Aussagekraft der Daten

Wie nützlich die gesammelten und ausgewerteten Daten sind, hängt vor allem von ihrer Aussagekraft ab. Daher ist es wichtig, an dieser Stelle die zugrundeliegenden Einschränkungen und Annahmen, des in dieser Arbeit entwickelten Prototypen, zusammenfassen.

- *Annahme des vollständigen Downloads:* Das entwickelte Verfahren zur Auswertung des Updatekommunikation beruht im Wesentlichen darauf, HTTP-GET-Requests bestimmter User-Agents herauszufiltern und anhand der URI festzustellen, ob und welches Softwarepaket angefordert wurde.
  Wie in *Abschnitt 5.3.1* beschrieben wird die Annahme gemacht, dass ein angefordertes Softwarepaket auch tatsächlich heruntergeladen und installiert wird. Ein Abbruch des Downloads oder der Installation kann nicht festgestellt werden. Auch ein HTTP-

GET-Request mit einer veralteten URI (Paket-Quelle ist umgezogen) kann nicht entdeckt werden, da keine HTTP-Statuscodes ausgewertet werden.
Auch wenn diese Fälle eher die Ausnahme darstellen, können also vom *Überwachungs-System* Einträge für Updates geführt werden, die gar nicht stattgefunden haben. Ebenso wenig kann festgestellt werden, ob ein Softwarepaket zwischenzeitlich wieder gelöscht wurde. Dies kann im Zweifel bei der Überprüfung auf Sicherheitslücken zu False-Positive-Meldungen führen, sprich, es wird eine Warnung für eine verwundbare Software auf einem Host herausgegeben, die jedoch gar nicht installiert ist.

- *Unsichere Aussagen durch generierte Softwarebezeichnungen:* Wie bereits in *Abschnitt 5.3.2* beschrieben ist das Generieren von alternativen Softwarebezeichnungen aus den URIs der mitgeschnittenen Update-Requests notwendig. Dadurch wird vermieden, dass durch das Fehlen von gemeinsamen Namenskonventionen bei Linux-Softwarepaketen und CPE-Namen Sicherheitslücken durch eine „inkorrekt“ abgeleitete Softwarebezeichnung übersehen werden.
  Durch diese alternativen Softwarebezeichnungen können aber unsichere oder gar inkorrekte Aussagen entstehen. So wurden unter den 362 Update-Einträgen, die im Rahmen dieser Evaluation aufgezeichnet wurden, bei 19 Einträgen Sicherheitslücken gemeldet. Dabei war bei einem Eintrag die Meldung inkorrekt und bei einem Weiteren eine genauere Untersuchung notwendig, wie nun im Folgenden aufgezeigt werden soll:
  Aus den drei mitgeschnittenen URIs

  - */ubuntu/pool/main/a/avahi/libavahi-client3_0.6.32%7erc%2bdfsg-1ubuntu2.2_amd64.deb*
  - */ubuntu/pool/main/a/avahi/libavahi-common3_0.6.32%7erc%2bdfsg-1ubuntu2.2_amd64.deb*
  - */ubuntu/pool/main/a/avahi/libavahi-common-data_0.6.32%7erc%2bdfsg-1ubuntu2.2_amd64.deb*

wurden die Softwarebezeichnungen „*avahi*“, „*libavahi*“, „*libavahi-common-data*“, „*libavahi-common3*“ und „*libavahi-client3*“ generiert und jeweils entsprechende Update-Einträge mit der Versionsnummer „0.6.32“ angelegt. *Avahi* ist ein Framework für die Multicast-DNS-Diensteerkennung [vgl. DEBIAN, 2018, AVAHI-DAEMON] und besteht aus mehreren Paketen, von denen die oben aufgeführten offensichtlich geupdated bzw. installiert wurden.
Eine Sicherheitslücke wurde nun aber nicht mit Hilfe einer der spezifischeren Softwarebezeichnungen festgestellt, sondern über den Namen des eigentlichen Frameworks, also „*avahi*“, wie in *Abbildung 6.5* dargestellt.

```
./m_terminal.py -id 126
CVE-2017-6519
CVE    : CVE-2017-6519
DATE   : 2017-04-30 21:59:00.297000
CVSS   : 6.4
avahi-daemon in Avahi through 0.6.32 inadvertently responds to IPv6
unicast queries with source addresses that are not on-link, which
allows remote attackers to cause a denial of service (traffic
amplification) or obtain potentially sensitive information via port-
5353 UDP packets. NOTE: this may overlap CVE-2015-2809.

References:
-----------
https://bugzilla.redhat.com/show_bug.cgi?id=1426712
https://www.secfu.net/advisories

Vulnerable Configs:
-------------------
cpe:2.3:a:avahi:avahi:0.6.32
```

Abbildung 6.5: CVE-Eintrag für „avahi“ in der Version „0.6.32“.

Die Beschreibung zu *CVE-2017-6519 legt* dar, dass die Sicherheitslücke im konkreten Paket mit dem Namen „*avahi-daemon*“ lokalisiert ist, welches während des Überwachungszeitraums jedoch nicht bezogen wurde. Die durch das *Überwachungs-System* gemachte Aussage, es sei ein

verwundbares Softwarepaket geupdated worden, ist somit streng genommen inkorrekt. Allerdings sollte an dieser Stelle ein Systemadministrator oder eine andere zuständige Person die gegebene Situation bewerten und feststellen, ob nicht dennoch der verwundbare *„avahi-daemon“* auf dem Host installiert ist, da die Versionsnummern der anderen mitgeschnittenen Avahi-Pakete darauf hinweisen könnten.
Des Weiteren sollte erwähnt werden, dass im Fall von *Avahi* eine CVE-Suche anhand von konkreten Paketnamen im CPE-Namen nur bedingt erfolgreich sein kann. Bei den in den CVE-Einträgen verwendeten CPE-Namen wird in der Regel „avahi“ nicht nur als *vendor* sondern auch als *product* geführt. Damit ist eine Zuordnung von Paketen anhand eines CPE-Namens nicht möglich [vgl. CVE-DETAILS, 2018, AVAHI].

- Ein weiterer Punkt, der erwähnt werden sollte und ebenfalls zu inkorrekten Aussagen führen kann, ist: Bei der Generierung von CPE-Namen für die Suche in der CVE-Datenbank wird das Hersteller-Attribut *vendor* nicht verwendet, sondern lediglich die Attribute *product* und *version*, da nur diese Informationen durch die Sensoren extrahiert werden. Dies kann zur Folge haben, dass es bei Softwareprodukten von verschiedenen Herstellern mit identischen Produktbezeichnungen zu Kollisionen kommen kann, die eine korrekte Auswertung auf Sicherheitslücken erschweren.

Die in diesem Abschnitt betrachtete *Aussagekraft der Daten* zeigt auf, inwiefern die angelegten Update-Einträge im *Überwachungs-System* die im überwachten Netzwerk eingesetzten Softwarepakete widerspiegeln. Unter Berücksichtigung der Annahmen und Einschränkungen des entwickelten Verfahrens kann mit dem Prototypen ein breites Spektrum an Softwarepaketen, die auf Linux-Systemen installiert oder aktualisiert werden, identifiziert und auf Sicherheitslücken hin untersucht werden, um verwundbare Systeme zu erkennen. Das Generieren von alternativen CPE-Namen kann gelegentlich dazu führen, dass durch eine Vereinfachung der Softwarepaketnamen False-Positive-Meldungen erzeugt werden, die manuell nachgeprüft werden müssen, wie es beispielsweise bei den

Softwarepaketen von *Avahi* der Fall war. Allerdings wird durch das Anbieten von alternativen CPE-Namen auch die Wahrscheinlichkeit verringert, dass Sicherheitslücken erst gar nicht erkannt werden.

7. Fazit

7.1 Zusammenfassung

Es wurde ein *prototypisches System* konzipiert und umgesetzt, welches unverschlüsselte Updatekommunikation von Linux-Systemen im Netzwerk erkennt und auswertet, um verwundbare Systeme zu identifizieren und zu visualisieren. Durch die Beobachtung des Netzwerkverkehrs während der Durchführung von Updates aber auch Neuinstallationen konnte ein Verfahren entwickelt werden, welches den Netzwerkverkehr nach bestimmten Mustern filtert, um angeforderte Softwarepakete zu erkennen und die Softwarebezeichnung und Versionsnummer zu extrahieren. Dies ermöglicht es darüber Rückschlüsse zu ziehen, welche Software auf welchem Host eingesetzt wird, um eine Überprüfung auf existente Sicherheitslücken durchführen zu können.
Dieses Verfahren wird in den *Sensor-Systemen* eingesetzt, die an verschiedenen Stellen eines Netzwerks platziert werden können, um dort den Netzwerkverkehr dezentral zu analysieren. Alle Updates und Neuinstallationen, die sie entdecken, werden an das zentrale *Überwachungs-System* übermittelt.
Das zentrale *Überwachungs-System* verwaltet alle durch die *Sensor-Systeme* zusammengetragenen Daten und ist zudem für die Untersuchung auf Sicherheitslücken zuständig. Für die Verwendung des *Überwachungs-Systems* wurden eine *Benutzer-* als auch eine *Programmierschnittstelle* entwickelt.
Die *Benutzerschnittstelle* wird in Form einer Kommandozeilenanwendung bereitgestellt und ermöglicht die Auswertung und Visualisierung der zusammengetragenen Daten in tabellarischer Form.
Die *Programmierschnittstelle* bietet die gleiche Funktionalität wie die Benutzerschnittstelle an. Alle Operationen geben ihre Ergebnisse im weit

verbreiteten JSON-Format zurück, um eine automatisierte Weiterverarbeitung durch andere Systeme zu ermöglichen.

Die in *Abschnitt 1.2* definierten Ziele dieser Abschlussarbeit konnten also vollständig erreicht werden. Das entwickelte Verfahren zur Auswertung der Updatekommunikation von Linux-Systemen zeigt, dass aus einer solchen passiven Analyse des Netzwerkverkehrs durchaus Nutzen gezogen werden kann. Die Informationen, die durch die Überwachung der Hosts eines Netzwerks gesammelt werden, können Aufschluss darüber geben, welche Software auf den Hosts eingesetzt wird. Dabei spielt es keine Rolle, ob es sich dabei um exponierte Dienste, am Netzwerkverkehr partizipierende Anwendungen oder ausschließlich lokal verwendete Anwendungen handelt. Natürlich müssen hierbei stets die zugrunde liegenden Annahmen und Einschränkungen des entwickelten Verfahrens berücksichtigt werden. So wird angenommen, dass die durch einen Host angeforderten Softwarepakete auch tatsächlich installiert werden. Es wird nicht berücksichtigt, ob der Download oder die Installation des Softwarepakets erfolgreich ist. Ebenso handelt es sich bei den identifizierten Neuinstallationen und Updates lediglich um Momentaufnahmen. Wenn im Laufe der Zeit Änderungen an den Hosts vorgenommen werden, wie beispielsweise das Entfernen von Anwendungen, sind diese für den Prototypen nicht erkennbar.
Auch können durch die Generierung von CPE-Namen falsche Aussagen bezüglich des Nicht- beziehungsweise Vorhandenseins von Sicherheitslücken entstehen. Durch diese Herangehensweise ist es aber auch möglich, ein breites Spektrum an Softwarepaketen auszuwerten, ohne hierfür ein gesondertes Verzeichnis zum Abbilden von Softwarepaket-Dateinamen auf CPE-Namen anlegen und verwalten zu müssen. Durch die Möglichkeit, eine Vielzahl von Softwarepaketen aus dem Stand identifizieren zu können, ist dieses Verfahren vor allem dann interessant, wenn die Sicherheitslücken in unscheinbaren Softwarepaketen lokalisiert sind und auch entdeckt werden.
Zuletzt sollte noch festgehalten werden, dass es sich bei dem hier entwickelten System um ein prototypisches System handelt, welches für die Integration in eine vorhandene IT-Infrastruktur noch weiterentwickelt werden sollte. So ist beispielsweise durch das unzuverlässige Sniffen

durch die Sensoren zu diesem Zeitpunkt nur eine lückenhafte Überwachung möglich. Auch sollte die manuelle Überprüfung der entdeckten Sicherheitslücken über die Kommandozeilenanwendung um weitere Möglichkeiten ergänzt werden, um den Arbeitsfluss zu optimieren.

7.2 Ausblick

Der entwickelte Prototyp bietet grundlegende Funktionen zur Überwachung von Linux-Systemen in einem Netzwerk an. Um einen wirkungsvollen Einsatz zu ermöglichen, sollte allerdings die in der *Evaluation* beschriebene Problematik des unzuverlässigen Sniffens behoben werden, da sonst zu viele Pakete nicht analysiert werden. Des Weiteren wäre es von Vorteil, wenn die Überwachung weitestgehend automatisiert werden könnte, sodass die Suche nach Sicherheitslücken nicht manuell angestoßen werden muss, sondern beispielsweise nach benutzerdefinierten Plänen durchgeführt wird.
Auch das automatisierte Generieren und Versenden von Warnungen bei entdeckten Sicherheitslücken würde die Reaktionszeit zur Behebung dieser verkürzen. Diese Warnungen könnten beispielsweise in Form von E-Mails an das zuständige Personal verschickt werden.
Eine grafische Benutzeroberfläche würde es erlauben, die Verwendung des *Überwachungs-Systems* intuitiver zu gestalten und neue Möglichkeiten in der Visualisierung der überwachten Linux-Systeme eröffnen. Durch die bereits existierende Programmierschnittstelle wäre die Einbindung einer solchen GUI leicht umsetzbar.

Abbildungsverzeichnis

Literaturverzeichnis

[Ehses et al., 2012] Erich Ehses, Lutz Köhler, Petra Riemer, Horst Stenzel, Frank Victor. Systemprogrammierung in UNIX / Linux. Wiesbaden: Vieweg+Teubner Verlag / Springer Fachmedien Wiesbaden GmbH, Wiesbaden, 2012

[Erben, 2017] Thomas Erben. Einführung in Unix/Linux für Naturwissenschaftler. Berlin: Springer Spektrum, 2017

[Foreman, 2009] Park Foreman. Vulnerability Management. CRC Press, 2009

[Gervasi, 2007] Computational Science and Its Applications - ICCSA 2007: International Conference, Kuala Lumpur, Malaysia, August 26-29, 2007. Proceedings, Part 2. Springer, 2007

[Greenbone, 2017] Greenbone Networks GmbH, Ralf Spenneberg, Alexander Rau. Greenbone Security Manager with Greenbone OS 4 User Manual. Greenbone Networks GmbH, 2017 *https://docs.greenbone.net/GSM-Manual/gos-4/en/GSM-Manual-GOS-4-en-20180404.pdf*

[Gregg/Haines, 2012] Michael Gregg, Billy Haines. CASP: CompTIA Advanced Security Practitioner Study Guide Authorized Courseware: Exam CAS-001. John Wiley & Sons, 2012

[Kappes, 2013] Markus Kappes. Netzwerk- und Datensicherheit. Wiesbaden: Springer Vieweg, 2013

[Kersten/Klett, 2012] Heinrich Kersten, Gerhard Klett. Der IT Security Manager. Wiesbaden: Springer Vieweg, 2012

[NIST, 2011] Brant A. Cheikes, David Waltermire, Karen Scarfone. NIST Interagency Report 7695, Common Platform Enumeration: Naming Specification Version 2.3. NIST, 2011
https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7695.pdf

[Patent US 7761918 B2, 2010] Ronald Joseph Gula, Renaud Marie Maurice Deraison, Matthew Todd Hayton. System and method for scanning a network. US 7761918 B2, 2010
https://encrypted.google.com/patents/US7761918

[Rohr, 2015] Matthias Rohr. Sicherheit von Webanwendungen in der Praxis. Wiesbaden: Springer Vieweg, 2015

[Steyer, 2018] Ralph Steyer. Programmierung in Python. Wiesbaden: Springer View, 2018

[Tenable, 2015] Passive Vulnerability Scanning Overview. Tenable Network Security Inc., 2015
https://www.tenable.com/sites/drupal.dmz.tenablesecurity.com/files/uploads/documents/whitepapers/PVS_Overview_0.pdf

[Wolfinger, 2013] Christine Wolfinger. Keine Angst vor Linux/Unix. Berlin: Springer Vieweg, 2013

[CSO, 2017, CVE-GAP] Closing the CVE gap: Is MITRE up to it?
https://www.csoonline.com/article/3204568/application-security/closing-the-cve-gap-is-mitre-up-to-it.html
(abgerufen am 03.07.2018)

[CVE-DETAILS, 2018, AVAHI]
https://www.cvedetails.com/vulnerability-search.php?f=1&vendor=&product=Avahi%2C+Avahi-daemon
(abgerufen am 10.09.2018)

[CVE-MITRE, 2018, ABOUT] About CVE
https://cve.mitre.org/about/index.html
(abgerufen am 07.06.2018)

[CVE-MITRE, 2018, CVE-2018-8885] CVE-2018-8885
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8885
(abgerufen am 07.06.2018)

[CVE-MITRE, 2018, NVD] CVE and NVD Relationship
https://cve.mitre.org/about/cve_and_nvd_relationship.html
(abgerufen am 03.07.2018)

[DEBIAN, 2018, AVAHI-DAEMON] Avahi mDNS/DNS-SD-Daemon
https://packages.debian.org/de/sid/avahi-daemon
(abgerufen am 05.09.2018)

[DISTROWATCH, 2018] DistroWatch Page Hit Ranking
https://distrowatch.com/dwres.php?resource=popularity
(abgerufen ab 08.10.2018)

[SCAPY, 2018, BUGS] *Known Bugs*
https://scapy.net/
(abgerufen am 03.09.2018)

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 _____ _____