Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Tim Eckhardt

Security-Monitoring von Internet-of-Things Endgeräten in lokalen Netzwerken unter Einsatz eines intelligenten Systems

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Tim Eckhardt

Security-Monitoring von Internet-of-Things Endgeräten in lokalen Netzwerken unter Einsatz eines intelligenten Systems

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. Stefan Sarstedt

Eingereicht am: 6. April 2017

Tim Eckhardt

Thema der Arbeit
Security-Monitoring von Internet-of-Things Endgeräten in lokalen Netzwerken unter Einsatz eines intelligenten Systems

Stichworte
Security-Monitoring, Internet der Dinge, IoT, intelligente Systeme, Machine Learning, Support Vector Machine, Knowledge Discovery, Anomalieerkennung

Kurzzusammenfassung
Aktuelle Angriffsereignisse zeigen, dass zentrale Sicherheitsaspekte im Internet-of-Things nicht immer ausreichend berücksichtigt werden und es daher essentiell ist, weiter an der IoT-Sicherheit zu forschen. In dieser Arbeit wird ein Konzept vorgestellt, wie Security-Monitoring von IoT Endgeräten in lokalen Netzwerken unter Einsatz eines intelligenten Systems betrieben werden kann. Die Implementierung wesentlicher Teile des Konzeptes zeigte gute Erkennungsraten und Weiterentwicklungsmöglichkeiten des Konzeptes.

Tim Eckhardt

Title of the paper
Security monitoring of internet of things devices in local networks using an intelligent system

Keywords
security monitoring, internet of things, IoT, intelligent systems, machine learning, support vector machine, knowledge discovery, anomaly detection

Abstract
Recent attacks show that crucial aspects of internet of things security are not always observed sufficiently. Therefore further research in the field of IoT security is of great importance. This thesis presents a concept of how security monitoring of IoT devices in local networks can be realised using an intelligent system. The implementation of substantial aspects of the concept shows good detection rates. Furthermore, the thesis points out recommendations for further research in the field of IoT security monitoring.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Der Begriff des Internet-of-Things (IoT) wurde erstmals durch Kevin Ashton im Jahre 1999 im Zusammenhang mit dem Versorgungskettenmanagement verwendet [Ash09]. Heute versteht man unter IoT ein Netzwerk von alltäglichen Gegenständen, welche ohne Abhängigkeit des Menschen Daten aus der echten Welt sammeln und auswerten können [Ash09; CLR10]. Die Anwendungsbereiche des IoT sind dabei sehr vielfältig und reichen vom Smart Home, über die Logistik bis hin zum Gesundheitsbereich [AIM10].

Unter anderem aufgrund dieser Vielfalt an Anwendungsmöglichkeiten, steigt die Zahl der IoT Geräte stetig an. Während es im Jahr 2016 weltweit rund 5 Milliarden vernetzte Geräte im IoT gab, soll diese Zahl bis ins Jahr 2020 auf über 20 Milliarden Geräte ansteigen [Sta17]. Diese Zahlen zeigen eindrucksvoll die steigende Relevanz des IoT auf und die rasante Entwicklung bringt es mit sich, dass zentrale Sicherheitsaspekte zum Teil nicht ausreichend berücksichtigt werden. So zeigen aktuelle Ereignisse die Anfälligkeit des IoT für Angriffsszenarien auf. Beispielsweise wurden im September 2016 DDos-Angriffe durch ein Botnet bestehend aus mehr als einer Million Geräte aus dem IoT durchgeführt. Dabei wurden Rekordwerte von bis zu 1,1 Terabit pro Sekunde gemessen [Ble17].

Um eine nachhaltige und sichere Entwicklung des IoT zu gewährleisten, ist es daher von essentieller Bedeutung, mögliche Abwehr- beziehungsweise Erkennungsmöglichkeiten solcher Angriffe zu erforschen. Die vorliegende Arbeit soll hierzu einen Beitrag leisten und beschäftigt sich mit der Frage, wie Security-Monitoring von IoT Endgeräten in lokalen Netzwerken unter Einsatz eines intelligenten Systems betrieben werden kann.

Hierfür wird ein Konzept zum IoT Security-Monitoring vorgestellt und in Teilen als Prototyp implementiert und evaluiert. Dazu werden in Abschnitt 2 zunächst ähnliche Forschungsarbeiten vorgestellt und es wird aufgezeigt, inwiefern sich die vorliegende Arbeit von diesen unterscheidet. Anschließend werden in Abschnitt 3 die für das behandelte Thema wesentlichen Grundlagen zu Internet-of-Things, Security-Monitoring und intelligenten Systemen dargestellt.

In Abschnitt 4 wird darauffolgend das Konzept für die Umsetzung einer Security-Monitoring Software auf Basis eines intelligenten Systems speziell für das Internet-of-Things vorgestellt. Abschnitt 5 zeigt die beispielhafte Implementierung von Teilen dieses Konzeptes anhand eines Prototypen. Anschließend werden in Abschnitt 6 die gewonnenen Ergebnisse präsentiert und in Abschnitt 7 diskutiert. Die Arbeit schließt in Abschnitt 8 mit einem Ausblick, wie auf Basis der gewonnenen Erkenntnisse das vorgestellte Konzept sowie zukünftige ähnliche Forschungsvorhaben sinnvoll erweitert werden könnten.

2 Related Work

Die vorliegende Arbeit beschäftigt sich mit der Fragestellung, wie ein Security-Monitoring mittels eines intelligenten Systems speziell für Internet-of-Things Endgeräte in lokalen Netzwerken umgesetzt werden kann. Um einen ersten Überblick über dieses Thema zu erhalten, werden nachfolgend ausgewählte ähnliche Forschungsarbeiten vorgestellt. Darüber hinaus wird die vorliegende Arbeit in diesen Rahmen eingeordnet.

In Bezug auf das Security-Monitoring im IoT-Bereich, liegt der Fokus häufig auf der Absicherung sowie der Standardisierung der Kommunikation wie mittels 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) und anderen [TCG+15; RNL11; SU16; May09]. Keoh et al. (2014) stellen in ihrer Arbeit verschiedene Standards gegenüber. Sie stellen fest, dass proprietäre Sicherheitslösungen auf lange Sicht problematisch für den Erfolg der Kommunikationssicherheit im IoT sind, da sie oftmals die Verwendung von sogenannten Middleboxes voraussetzen und somit die Komplexität und die Kosten bei der Verwendung der IoT Geräte erhöhen [KKT14]. Als Alternative für eine nahtlose Verbindung zwischen verschiedenen Geräten stellen sie den von der IETF[1] entwickelten Standard 6LoWPAN vor, dem sie eine kritische Rolle bei der Absicherung der IoT zuschreiben. Neben der Absicherung und Standardisierung der Kommunikation im IoT, konzentrieren sich andere Sicherheitsansätze auf das Aufspüren von Botnetzen und deren Command and Control (C&C) Server im Internet, welche das IoT im besonderen Maße bedrohen. Luz (2013) setzt beispielsweise darauf, durch das passive Sammeln von DNS-Anfragen, dem sogenannten pDNS (passive DNS), Veränderungen der DNS History von Domains zu erfassen und diese mit Hilfe von Machine Learning zu analysieren [Luz13]. Eine Annahme besteht darin, dass Domains mit Malwarebezug kürzere Time to Live (TTL) Werte aufweisen als Domains ohne Malwarebezug. Mit den erfassten pDNS Daten von zwei Wochen sowie der Verwendung von Blacklists war Luz (2013) durch den Einsatz von verschiedenen Machine Learning Klassifikatoren in der Lage, eine Precision (Genauigkeit) von 97% und eine False Positive Rate von 3% zu erreichen.

[1] Internet Engineering Task Force

Die Methode des Security-Monitorings und der Intrusion Detection existiert ebenfalls bereits seit einiger Zeit und wird ausgiebig erforscht [Raz13; FSME15]. Für den Ansatz des spezifisch für IoT ausgelegten Security-Monitorings, wie es Thema der vorliegenden Arbeit ist, existieren meines Wissens nach bisher jedoch nur wenige Arbeiten. Ein Ansatz in diesem Bereich ist SVELTE, ein real-time Intrusion Detection System für das IoT [RWV13]. SVELTE ist dabei vorrangig für die Verwendung von RPL (Routing Protocol for Low-Power and Lossy Networks) basierten 6LoWPAN Netzwerken entwickelt worden. Der Schwerpunkt der Angriffserkennung dieses IDS liegt im Gegensatz zu der vorliegenden Arbeit auf der Erkennung von Routing-Attacken wie dem Sinkhole, Selective Forwarding sowie Spoofed und Altered Information. Der technische Hintergrund der Angriffserkennung von SVELTE ist die Verwendung eines Hybriden Systems, mit einer Kombination aus Signatur- und Anomalieerkennung. Weiterhin setzt SVELTE das Contiki OS, ein für das IoT entwickeltes Betriebssystem, auf den Endgeräten voraus. Bei verschiedenen Testkonfigurationen erreichte SVELTE true-positive Raten zwischen 90% und 100%. Dasgupta (1999) beschreibt einen weiteren Ansatz für ein Intrusion Detection System, welches an das Prinzip des menschlichen Immunsystems angelehnt ist [Das99]. Dabei kommen sogenannte Immunity-Bases Agents zum Einsatz, welche auf den einzelnen Geräten eingesetzt werden und den aktuellen Zustand des Systems beobachten. Sie suchen dabei nach Fehlern, Auffälligkeiten, Eindringlingen und ähnlichem. Diese Agents sind außerdem in der Lage, sich untereinander über ihre Aktivitäten auszutauschen. Darüber hinaus können sie ihre Aktionen durch Machine Learning an Sicherheitsrichtlinien und die jeweilige Umgebung anpassen. Sie sind damit in der Lage, bekannte und unbekannte Einbrüche in ein System zu erkennen. Dasguptas Ansatz ist nicht im Hinblick auf das IoT entwickelt worden. Die zugrunde liegenden Konzepte lassen sich jedoch auf das IoT übertragen. Auch der sogenannte One-Class SVM wurde bereits im Bereich der intrusion detection eingesetzt. So erzielten Wang et al (2004) mittels One-Class SVM bessere Genauigkeiten bei der intrusion detection als mit anderen Kernel-basierten Anomalieerkennungsverfahren [WWM04].

Zusammenfassend lässt sich sagen, dass es verschiedene Ansätze und Herangehensweisen gibt, sich mit der Absicherung des IoT auseinander zu setzen. In Bezug auf die vorliegende Arbeit lässt sich diesbezüglich festhalten, dass das in Abschnitt 4 dargestellte Konzept Gemeinsamkeiten zu einigen vorgestellten Arbeiten aufweist. So wird beispielsweise Machine Learning, darunter auch die One-Class SVM, zur Anomalieerkennung genutzt. Gleichzeitig zeigen sich Unterschiede zu den vorgestellten Arbeiten. So setzt das Konzept im Gegensatz zu SVELTE kein bestimmtes Betriebssystem auf den Endgräten voraus. Außerdem ist es im Unterschied zur Herangehensweise von Dasgupta nicht agentenbasiert. Das in Abschnitt 4 erläuterte Konzept

baut also auf vorhandenen Arbeiten auf und integriert gleichzeitig wenig erforschte und neue Herangehensweisen.

3 Grundlagen

Wie bereits einleitend erwähnt wurde, geht es in der vorliegenden Arbeit um die Entwicklung eines Konzeptes zum Security-Monitoring von IoT Endgeräten mittels eines intelligenten Systems. Nachdem in Abschnitt 2 bereits auf Gemeinsamkeiten und Unterschiede zu ähnlichen Forschungsvorhaben eingegangen wurde, werden im Folgenden die für die vorliegende Arbeit wesentlichen Grundlagen dargestellt. Hierzu werden neben dem Internet-of-Things auch das Security-Monitoring sowie die intelligenten Systeme näher erläutert.

3.1 Internet-of-Things

Um einen Überblick über wesentliche Aspekte des Internet-of-Things zu geben, wird der Begriff im Folgenden definiert und seine Entwicklung skizziert. Darüber hinaus werden die technischen Grundlagen des IoT dargestellt.

3.1.1 Definition und Entwicklung

Um den Begriff des Internet-of-Things (IoT) zu erklären, ist es zunächst sinnvoll, die Entwicklung des Internets zu betrachten. Es zeigt sich dabei, dass das Internet seit seinem Bestehen einem fortwährenden Wandel unterliegt, der in Abbildung 3.1 überblicksartig dargestellt ist.

Wo zu Beginn nur statische HTML Dokumente miteinander verlinkt wurden und somit die Grundlage des World Wide Web (WWW) bildeten, rückte nach einiger Zeit der Wunsch nach Interaktion und Zusammenarbeit in den Vordergrund. Aus dem frühen WWW entwickelte sich mit dem Aufkommen der Datenbank gestützten Systeme, wie der Content Management Systeme (CMS), das sogenannte Web 2.0, wodurch das dynamische Austauschen und Ergänzen von Inhalten möglich wurde. Im weiteren Verlauf der Entwicklung ging der Trend über soziale Medien hin zum Internet-of-Things, wobei sich diese Entwicklungen gegenseitig ergänzen und

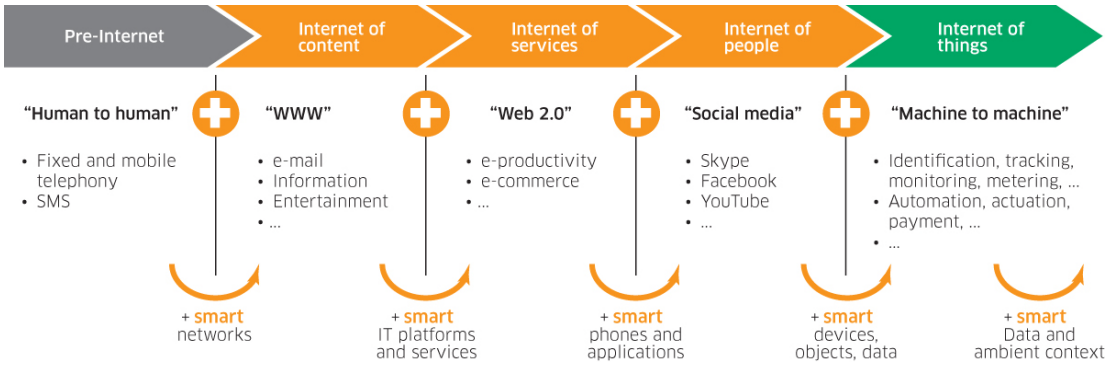

Abbildung 3.1: Entwicklung des Internets [nach Jad15]

nicht notwendigerweise ersetzen [WAX15].

In Bezug auf das Internet-of-Things liegt nach wie vor keine einheitliche Begriffsdefinition vor. Erstmals wurde der Begriff durch Kevin Ashton im Jahre 1999 im Zusammenhang mit dem Versorgungskettenmanagement verwendet [Ash09; GBMP13]. Ashton beschreibt das IoT als Netzwerk von alltäglichen Gegenständen, welche ohne Abhängigkeit des Menschen Daten aus der echten Welt sammeln und auswerten können [Ash09; CLR10]. Dabei werden diese Gegenstände beispielsweise mit Sensoren (z.B. für Temperatur, Licht, Wasserverbrauch, Erschütterungen) und/ oder Aktoren (z.B. Netzschalter, Motoren) ausgestattet [CLR10]. Die auf diese Weise ausgestatteten Gegenstände, werden mittels kabelgebundenen oder kabellosen Netzwerken verbunden. Durch diese Art der Vernetzung, erhalten die Gegenstände über ihren ursprünglichen Zweck hinaus einen digitalen Nutzen. Mattern (2010) schreibt hierzu: „*Using sensors, [smart objects] are able to perceive their context, and via built-in networking capabilities they would be able to communicate with each other, access Internet services and interact with people.* Digitally upgrading *conventional object in this way enhances their physical function by adding the capabilities of digital objects, thus generating substantial added value.*“ [MF10, S. 242–243]. Die Anwendungsgebiete des IoT sind mittlerweile sehr vielseitig und reichen vom Verkehrswesen, über die Logistik und den Gesundheitsbereich bis hin zum *intelligenten Zuhause* [AIM10; LDXZ15].

Durch die Vernetzung von Gegenständen im IoT, folgt auch eine Veränderung der digitalen Kommunikation insgesamt. Ashton (2009) stellt hierzu fest: „*Today computers— and, therefore, the Internet— are almost wholly dependent on human beings for information. Nearly all of the*

[...] data available on the Internet [was] first captured and created by human beings— by typing, pressing a record button, taking a digital picture or scanning a bar code.“ [Ash09, S. 1]. So findet auch die Kommunikation im Internet bisher hauptsächlich von *Mensch zu Mensch* statt [TW10]. In der näheren Zukunft wird hingegen so gut wie jedes Objekt vernetzt werden können. Durch die Anzahl vernetzter Dinge wird auch der durch sie generierte Traffic stark steigen und den Großteil des gesamten Traffics ausmachen [IBM09]. Wenn zunehmend auch die kleinsten Objekte vernetzt sein werden, wird sich die Kommunikation von *Mensch zu Mensch* hin zu *Mensch zu Maschine* und *Maschine zu Maschine* verlagern [TW10].

Diese Veränderung der digitalen Kommunikation durch das IoT kann verschiedene Vor-, aber auch Nachteile mit sich bringen. So können durch diese Art der Kommunikationsveränderung aufgrund von Vernetzung perspektivisch wirtschaftliche Vorteile generiert werden. Ashton (2009) schreibt hierzu: „*If we had computers that knew everything there was to know about things — using data they gathered without any help from us — we would be able to track and count everything, and greatly reduce waste, loss and cost.*“ [Ash09, S. 1]. Andererseits kann die vermehrte Nutzung von IoT auch Nachteile mit sich bringen, die insbesondere aufgrund von Sicherheitslücken auftreten können. Auf diese Aspekte wird in Punkt 3.2.2 näher eingegangen.

3.1.2 Technische Grundlagen

Die technischen Grundlagen des IoT sind sehr vielschichtig. Li et al. (2015) stellen hierzu fest: „*The words* Internet *and* Things *mean an inter-connected world-wide network based on sensory, communication, networking, and information processing technologies [...]*“[LDXZ15, S. 244]. Die Abbildung 3.2 stellt die vielschichtige Architektur des IoT grafisch dar.

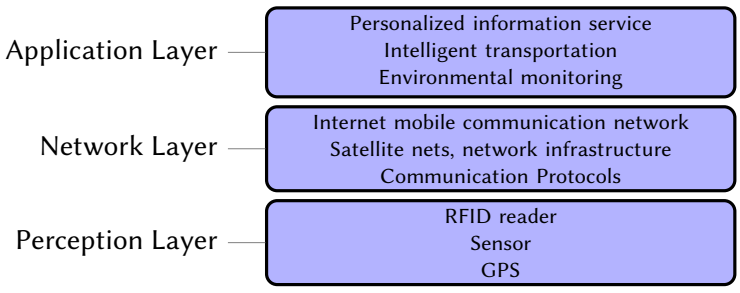

Abbildung 3.2: IoT Architektur [nach SWZL12, S. 648]

Nachfolgend werden die einzelnen Layer überblicksartig dargestellt. Der Fokus liegt dabei auf dem Network Layer, da dieser für die vorliegende Arbeit von besonderer Relevanz ist.

Perception Layer

Der Perception Layer umfasst alle IoT Geräte, die Informationen aus der Umwelt sammeln und/oder mit dieser interagieren. Er beinhaltet also alle mit Sensoren und/oder Aktoren ausgestatteten und vernetzten Geräte. Die Identifizierung von Objekten in der Umwelt kann beispielsweise mittels Radio Frequency Identification (RFID) erfolgen.

Network Layer

Der Network Layer ist dafür zuständig, die vom Perception Layer gesammelten Informationen zum jeweiligen Verarbeitungssystem zu übertragen. Er übernimmt dabei die Aufgaben der Adressierung, Weiterleitung, des Routing sowie der Einhaltung der Sicherheitsprotokolle. Dabei werden häufig bereits bestehende Kommunikationsnetzwerke, wie das Internet, Mobilnetze und Andere verwendet. Das Transportmedium, der Versand und das Format variieren je nach Anwendungsfall sehr stark. Aufgrund dieser Variation, werden nachfolgend einige ausgewählte Aspekte beispielhaft dargestellt.

In Bezug auf die Kommunikationstechnologien wurde im IoT-Bereich zunächst insbesondere RFID eingesetzt [Ash09; LDXZ15]. Im Laufe der Zeit kamen weitere hinzu beziehungsweise wurden in den IoT-Sektor übertragen, um neuen Anforderungen (z.B. Transport größerer Datenmengen, höhere Übertragungsreichweite) gerecht zu werden. Mittlerweile gibt es im IoT neben RFID eine Vielzahl weiterer Technologien und Standards wie beispielweise NFC, ZigBee, Bluetooth Low Energy (BLE) und WiFi, die sich beispielsweise hinsichtlich der Datenübertragungsrate, der Bandbreite sowie dem Übertragungsbereich unterscheiden [FT02; LDXZ15]. Die Abbildung 3.3 gibt einen Überblick über diese und weitere Kommunikationstechnologien des IoT.

Eine Standardisierung dieser im IoT Bereich genutzten Kommunikationstechnologien erweist sich als eine große Herausforderung. Dies liegt unter anderem darin begründet, dass es im IoT eine Vielzahl möglicher Endgeräte gibt, die sich hinsichtlich Größe, Form und Anwendungsbereich zum Teil deutlich unterscheiden. So sind einige Endgeräte sehr klein, um beispielsweise in Kleidung eingenäht zu werden und dürfen nur sehr wenig Strom verbrauchen. Andere haben deutlich mehr Rechenkraft und müssen in der Lage sein, größere Datenmengen übertragen zu können. Bisher werden daher meist spezifische Technologien für den jeweiligen Anwendungsfall genutzt.

Kommunikationstechnologie	Datenübertragungsrate	Bandbreite	Übertragungsbereich
RFID	424 kbps	135 KHz	>50 cm
		13.56 MHz	>50 cm
		866–960 MHz	>3 m
		2.4 GHz	>1.5 m
NFC	100 kbps–10 Mbps	2.45 GHz	
ZigBee	256 kbps/20 kbps	2.4 GHz/900 MHz	10 m
Bluetooth	1 Mbps	2.4 GHz	10 m
BLE	10 kbps	2.4 GHz	10 m
UWB	50 Mbps	Wide range	30 m
WiFi	50–320 Mbps	2.4/5.8 GHz	100 m
Wi-Max	70 Mbps	2–11 GHz	50 km
UMTS/CDMA/EDGE/MBWA	2 Mbps	896 MHz	∼

Abbildung 3.3: Auswahl von Kommunikationstechnologien im IoT [nach LDXZ15, S. 251]

Allen IoT Endgeräten gemeinsam ist jedoch, dass sie jeweils Teil eines Netzwerkes von miteinander verbundenen IoT Knotenpunkten sind. Wie bereits in Punkt 3.1.1 erwähnt, kann diese Verbindung sowohl kabelgebunden, als auch kabellos sein, wobei die meisten aktuellen IoT Endgeräte eine drahtlose Anbindung unterstützen. Die drahtlose Anbindung kann dabei auf unterschiedlichen Protokollen basieren. Bei leistungsstärkeren Geräten, wie auch bei der in dieser Arbeit verwendeten IP-Kamera, wird oftmals IEEE 802.11 (WLAN) verwendet, welches das Internet Protocol (IP) nutzt [CLR10]. Im bisher noch gängigen IPv4 können maximal 2^{32} (4.294.967.296) Adressen vergeben werden, wobei jeder Knoten über eine 4-Byte große Adresse identifiziert wird. Durch den enorm großen Zuwachs an Endgeräten verringert sich die Anzahl der noch verfügbaren Adressen sehr schnell, so dass der Adressraum im IPv4 bald vollständig belegt sein wird. Die relativ geringe Anzahl zu vergebender Adressen stellt daher einen technisch limitierenden Faktor des IPv4 dar und führt dazu, dass mittlerweile eine Umstellung auf IPv6 stattfindet. Mit IPv6 ist es möglich, 2^{38} Adressen zu definieren, was auf lange Zeit ausreichen sollte, alle Geräte eindeutig zu adressieren [AIM10; GBMP13]. Aufbauend auf IPv6 wird an einem 6LoWPAN (IPv6 over low-power Wireless Personal Area Network) Protokoll gearbeitet, was unter anderem der Verringerung von zu übertragenden Datenmengen und der möglichst einfachen Einbindung auch kleinster Endgeräte in bestehende Netze dienen soll. In dieser Arbeit wird IPv4 genutzt, da die Hardware des Testaufbaus dies voraussetzt.

Application Layer

Der Application Layer ist die oberste und damit abschließende Schicht der IoT Architektur. Hier findet die Verarbeitung und die Darstellung der Daten speziell für den Nutzer statt. Die Daten werden für den jeweiligen Anwendungszweck weiterverarbeitet und synchronisiert. Dem Nutzer kann dann eine grafische Benutzeroberfläche (GUI) zur Verfügung gestellt werden, um auf die angebotenen Dienste und die aufbereiteten Daten zugreifen zu können [SU16].

In Bezug auf die technischen Grundlagen des IoT lässt sich also festhalten, dass die Vielzahl an Endgeräten und Anwendungsbereichen zu einer ebensolchen Vielfalt an Technologien führt.

3.2 Security-Monitoring

Unter Security-Monitoring versteht man *„[…] the collection, analysis, and escalation of indications and warnings to detect and respond to intrusions.“*[Bej04]. Um die für die vorliegende Arbeit wesentlichen Aspekte des Security-Monitorings darzustellen, soll zunächst ein Überblick über die IT- und Informationssicherheit im Allgemeinen gegeben werden. Anschließend sollen die für das IoT relevanten Sicherheitsaspekte näher betrachtet werden. Der Security-Monitoring Teil schließt mit einem Abschnitt zur Intrusion Detection, die einen Teilbereich des Security-Monitorings darstellt und als Teil dieser Arbeit genutzt wurde, um infizierte IoT Endgeräte in lokalen Netzwerken mittels Machine Learning zu erkennen.

3.2.1 IT-Sicherheit und Informationssicherheit

Die IT-Sicherheit im Allgemeinen bezeichnet den Schutz vor Angriffen auf IT-Systeme, welche aufgrund von Bedrohungen und Schwachstellen Risiken ausgesetzt sind. Die Informationssicherheit wiederum *„[…] hat den Schutz von Informationen als Ziel. Dabei können Informationen sowohl auf Papier, in Rechnern oder auch in Köpfen gespeichert sein. IT-Sicherheit beschäftigt sich an erster Stelle mit dem Schutz elektronisch gespeicherter Informationen und deren Verarbeitung. Der Begriff Informationssicherheit statt IT-Sicherheit ist daher umfassender und wird zunehmend verwendet.“*[BSI17].

Die wesentlichen Schutzziele der Informationssicherheit betreffen den Erhalt von Vertraulichkeit (engl. confidentiality), Integrität (engl. integrity) und Verfügbarkeit (engl. availability).

Abbildung 3.4: Wesentliche Schutzziele der Informationssicherheit

Unter Vertraulichkeit versteht man das Ziel, dass nur autorisierte Personen beziehungsweise Einheiten Zugriff auf die Informationen erhalten. Integrität meint, dass die Informationen korrekt und vollständig sind. Verfügbarkeit wiederum bezieht sich darauf, dass die Informationen für autorisierte Personen beziehungsweise Einheiten zugänglich und nutzbar sind [PA10].

3.2.2 Sicherheitsaspekte im IoT

Bei der Vielzahl von IoT-Produkten, gibt es immer auch Solche, bei denen die Schutzziele der Informationssicherheit nicht ausreichend berücksichtigt werden und dadurch Sicherheitsprobleme entstehen.

Da das Internet-of-Things auf der Technologie des Internets basiert, hat es auch ähnliche Sicherheitsprobleme wie das Internet selbst. Die Abbildung 3.5 zeigt eine Auswahl möglicher Gefahren der Sicherheit in Bezug auf die IoT-Layerstruktur nach Jing (2014)[JVW+14].

In der vorliegenden Arbeit wird vorrangig die Erkennung einer kompletten Übernahme von IoT Geräten durch eine Schadsoftware betrachtet. Nachfolgend wird ein mögliches Angriffsszenario für eine Übernahme eines IoT Geräts auf dem Application Layer beschrieben, so wie es auch in der vorliegenden Arbeit durchgeführt wurde, um Testdaten für die Erkennung solcher Angriffe zu gewinnen. Dabei wird angenommen, dass es im Internet eine Reihe von Endgeräten gibt, die sehr schwache Passwörter zur Absicherung des Zugangs nutzen. Die schwachen Passwörter bestehen entweder bereits im Auslieferungszustand des Gerätes oder sind nachträglich vom Benutzer festgelegt worden. Stout und Urias (2016) stellen dazu fest:

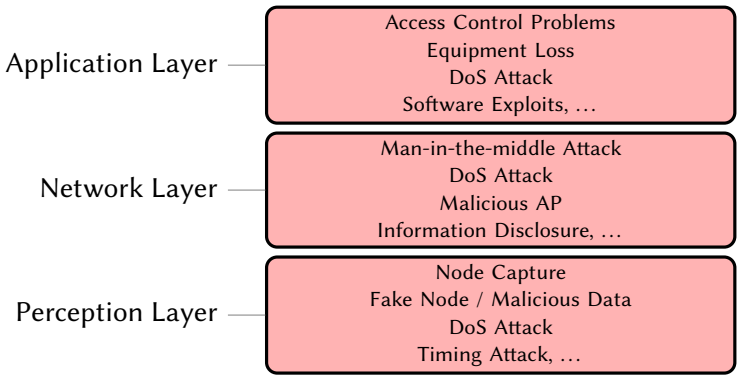

Abbildung 3.5: Auswahl von Sicherheitsaspekten im IoT [nach JVW+14, S. 2484–2485]

„*Vulnerabilities caused by using simple passwords or relying on default passwords on embedded systems can be easily exploitable. Furthermore, such devices are often never powered down; persistent network connectivity effectively shortens the attack time against vulnerabilities of largely unsecured end point devices.*“[SU16, S. 6].

Einige Angriffsmethodiken nutzen diese schwachen Passwörter aus, indem sie automatisiert Anmeldeversuche auf zufälligen Adressen im Internet durchführen. Bei diesen Anmeldeversuchen kommen oftmals Passwortlisten zum Einsatz, welche eine Reihe von bekannten Standardpasswörtern enthalten. Diese werden der Reihe nach ausprobiert und bei einer erfolgreichen Anmeldung wird beispielsweise eine Schadsoftware installiert, welche dann für verschiedene, oftmals böswillige Zwecke eingesetzt wird. So hat der Angreifer direkten Zugriff auf das lokale Netzwerk, in dem sich das System befindet und kann beispielsweise private Informationen auslesen und weitere im Netzwerk befindliche Systeme angreifen. Weiterhin kann sich das System mit anderen infizierten Systemen vernetzen, um verteilte Angriffe auf ein Zielsystem durchzuführen, sogenannte DDos-Angriffe. Eine weitere Form der Bedrohung besteht in gezielten Angriffen auf Systeme von denen ein Angreifer bespielsweise bestimmte Informationen bekommen möchte. Man spricht in solchen Fällen auch oftmals von einem APT (Advanced Persistent Threat), also einer über einen längeren Zeitraum andauerden Bedrohung, wobei der Angreifer möglichst lange versucht, ein System auszuspionieren.

Es bleibt festzuhalten, dass es im IoT Bereich zahlreiche Sicherheitsaspekte gibt, die berücksichtigt werden müssen, um die Schutzziele der Informationssicherheit gewährleisten zu können. Das vorgestellte mögliche Angriffsszenario zeigt darüber hinaus beispielhaft eine Schwachstelle im IoT auf, die auch ein Einfallstor für weiterführende Angriffe darstellen

kann. In der vorliegenden Arbeit soll nachfolgend gezeigt werden, wie solche und ähnliche Angriffsszenarios durch ein Security-Monitoring System erkannt werden können.

3.2.3 Intrusion Detection System

Um einige der in 3.2.2 dargestellten Angriffe erkennen zu können, kann ein Intrusion Detection System (IDS) verwendet werden. Ein IDS ist Teil des Security-Monitorings. Hierbei werden Daten gesammelt und anschließend analysiert, um mit den Ergebnissen Angriffe auf ein IT-System zu erkennen und daraufhin eine Warnung zu erzeugen. Das Ziel eines IDS ist es, eine zu 100% genaue Erkennungsrate für Angriffe zu ermöglichen und gleichzeitig Falschalarme vollständig zu vermeiden. Dies kann allerdings aufgrund der Komplexität der Einflussfaktoren nicht gewährleistet werden. Es kann also dazu kommen, dass einige Vorgänge in den gesammelten Daten nicht als Einbruch erkannt werden und somit auch keine Warnung ausgegeben werden kann. In diesem Fall spricht man von einer false-negative Klassifikation. Wird hingegen fälschlicherweise ein Vorgang als Einbruch eingestuft, obwohl es sich dabei nicht um einen Solchen handelt, spricht man von einer false-positive Klassifikation.

Aufzeichnung der Aktivitäten

Das Intrusion Detection System kann je nach Aufgabengebiet unterschiedliche Schwerpunkte haben und so unterscheiden sich auch die analysierten Daten je nach Anwendungsfall. Liegt der Fokus auf der Überwachung eines Systems, wie beispielsweise eines Webservers, werden hier die Logdateien des Apache-HTTP-Servers und der Datenbank bevorzugt analysiert. Dabei handelt es sich dann um ein so genanntes **Host Intrusion Detection System** (HIDS). In anderen Fällen soll hingegen der Datenverkehr eines Netzes analysiert werden, wobei nicht auf bestehende Logdateien der einzelnen Systeme zurückgegriffen wird. In diesem Fall kommt ein **Network Intrusion Detection System** (NIDS) zum Einsatz, welches versucht, alle Netzwerkpakete aufzuzeichen, um diese dann für die Analysevorgänge zu verwenden.

Erkennungsmethoden

Ein IDS kann die Host- oder Netzwerkbasierten Aktivitäten traditionell mit zwei unterschiedlichen Erkennungsmethoden auswerten. Zum einen können die Daten mittels **Sig-**

naturerkennung mit bereits bekannten Angriffen verglichen werden. Dabei wird versucht, ein Muster in den Daten zu finden, welches mit solchen in einer Datenbank bekannter Angriffe übereinstimmt. Hierbei können nur bereits bekannte Angriffsmuster erkannt werden. Im Gegensatz dazu steht die Methode der **Anomalieerkennung** [RWV13]. Die Anomalieerkennung versucht, Abweichungen von der Norm, sogenannte Anomalien zu erkennen. Barnett und Lewis (1994) definieren eine Anomalie als „*[...] an outlier in a set of data to be an observation (or subset of obervations) which appears to be inconsistent with the remainder of that set of data.*“[BL94, S.7] Die Norm wiederum spiegelt das typische Verhalten des Systems wider und es wird angenommen, dass sich das System in Folge eines Angriffs anders verhält als bisher. Als technische Grundlage für die Anomalieerkennung wird häufig Machine Learning (3.3.2) verwendet, so wie es auch im Rahmen dieser Arbeit zum Einsatz kommt.

3.3 Intelligente Systeme

Nachdem in den vorangegangenen Abschnitten bereits auf Sicherheitsaspekte im IoT sowie Intrusion Detection Systeme eingegangen wurde, erfolgt in den folgenden Abschnitten zunächst eine Definition des Begriffs der intelligenten Systeme. Anschließend wird das Machine Learning näher erläutert, welches im Rahmen der vorliegenden Arbeit als Grundlage für die Anomalieerkennung eingesetzt wird. Hierbei wird detaillierter auf die Support Vector Machine, SciPy und Scikit-Learn sowie den Knowledge Discovery Process eingegangen, da diese Aspekte wesentlich für das in der vorliegenden Arbeit vorgestellte Konzept sind.

3.3.1 Definition

Bisher liegt noch keine allgemeingültige Definition für den Begriff des intelligenten Systems vor. Dies ist unter anderem darin begründet, dass schon der Begriff der Intelligenz bisher nicht eindeutig definiert ist. Man kann jedoch festhalten, dass man unter einem intelligenten System ein technisches System versteht, welches zu „intelligentem“ Verhalten fähig ist. Die Grundlage für intelligente Systeme bildet die Forschung aus dem Bereich der Künstlichen Intelligenz. Hierbei wird versucht, ein System so zu programmieren, dass es eigenständig zur Lösung von Problemen beitragen kann.

3.3.2 Machine Learning

Das Machine Learning (Maschinelles Lernen) ist ein Teilbereich der intelligenten Systeme und ist darauf spezialisiert, künstliches Wissen mittels eines Lernprozesses aus Daten zu gewinnen. Man kann also davon sprechen, dass durch dieses Verfahren eine Art künstlicher Erfahrungsschatz aufgebaut wird, auf welchen bei der Auswertung zukünftiger Daten zurückgegriffen wird [MST94]. Machine Learning teilt sich grob in die Ansätze des **Supervised Learning** (Überwachtes Lernen), des **Unsupervised Learning** (Unüberwachtes Lernen) und des **Reinforcement Learning** (Bestärkendes Lernen). Diese Ansätze unterscheiden sich grob durch die genutzten Trainingsdaten und den Prozess des Lernens.

Supervised Learning

Beim Supervised Learning geht der eigentlichen Voraussage (engl. Prediction) immer eine Lernphase mit Trainingsdaten voraus. Hierbei werden die Trainingsdaten so vorbereitet, dass die zu lernende Ausgabe durch einen Experten mit in den Lernvorgang einfließt. Somit kennt das System bereits in der Lernphase die korrekte Ausgabe für jeden Trainingsdatensatz. Als Analogie zum menschlichen Lernen ist das vergleichbar mit einem Lehrer, der einem den Lehrstoff nahebringt oder mit dem Lesen eines gut strukturierten Fachbuchs zu einem speziellen Thema.

Unsupervised Learning

Bei der Methode des Unsupervised Learning hingegen, kennt das System in der Lernphase die gewünschten Ausgabewerte nicht im Voraus. Es wird also versucht, ein Muster in den Lerndaten zu erkennen und dieses dann auf die zukünftigen Voraussagen anzuwenden. Unsupervised Learning ist also ein Lernen durch Beobachtung und Entdeckung, da es keinen *Lehrer* gibt, welcher einem die korrekten Muster oder Klassifizierungen aufzeigt. Carbonell (1983) schreibt dazu „*This is a very general form of inductive learning that includes discovery systems, theory-formation tasks, the creation of classification criteria to form taxonomic hierarchies, and similar tasks without benefit of an external teacher*“ und weiter „*The learner is not provided with a set of instances of a particular concept, nor is it given access to an oracle that can classify internally-generated instances as positive or negative instances of any given concept.*“ [CMM83, S. 10].

Reinforcement Learning

Das Reinforcement Learning setzt wiederum auf ein Belohnungssystem. Dabei gibt ein Experte verschiedene Belohnungen für einzelne erzielte Erfolge vor und das System versucht durch immer neue Durchläufe, diese Belohnungen zu maximieren. In Analogie zum menschlichen Lernen könnte man hier von *Learning by Doing* sprechen, welche im lebenslangen Lernprozess eine der häufigsten Lernformen ist.

Neben der Einteilung der Machine Learning Ansätze in Gruppen mit Fokus auf dem Lernvorgang, kann auch eine Klassifikation ausgehend von der Art der produzierten Ausgaben stattfinden. Auf diese Weise lassen sich die Ansätze grob in die Kategorien Classification, Regression und Clustering einteilen.

Classification

Bei der Classification werden die Ausgaben in zwei oder mehr Klassen eingeteilt [MST94]. Ein typisches Beispiel für dieses Vorgehen ist die Einteilung von E-Mails in die Klassen „Spam“ und „kein Spam“. Für die Umsetzung der Klassifikation stehen verschiedene Algorithmen zur Verfügung. Einige Beispiele sind Support Vector Machines (SVM), Kernel Estimation, Neural Networks, Decision Trees und Linear Classifiers.

Regression

Der Ansatz des Regression Modells bezieht sich auf die fortwährenden Ausgaben. Hier findet also keine Einteilung in Klassen wie bei der Classification statt, sondern es werden laufende Zahlen für die Vorhersage ausgegeben. Ein Beispiel für dieses Modell ist die Prognose von Mietpreisen für Wohnungen bestimmter Größe innerhalb eines Stadtgebietes. Algorithmen für diesen Einsatz sind beispielsweise Support Vector Regression (SVR) und Linear SVR.

Clustering

Clustering versucht, Objekte anhand ihrer Werte zu gruppieren, so dass ähnliche Objekte in die gleichen Gruppen gelangen und unähnliche Objekte in andere Gruppen eingeteilt wer-

den [FGK+10]. So findet eine Art Klassifizierung statt, ohne die Klassen vorher zu kennen. Beispiele für Clustering Algorithmen sind K-Means, Spectral Clustering und Affinity Propagation.

3.3.3 Support Vector Machine

Die Support Vector Machine wurde bereits als einer der möglichen Algorithmen im Bereich der Classification angeführt und soll hier näher erläutert werden.

Unter sogenannten Support Vector Machines (SVMs) versteht man eine Sammlung von Lerntechniken. Bei allen diesen Lerntechniken wird auf Basis von Trainingsdaten — zumeist einem Paar von „input objects“ (häufig Vektoren) und dem gewünschten Output — eine Funktion erzeugt. Diese gelernte Funktion kann dann genutzt werden, um den Output für neue Objekte vorherzusagen [Yu09; Ali09]. Zusammenfassend kann man eine SVM definieren als: „*[…] a computer algorithm that learns by example to assign labels to objects. […] In essence, an SVM is a mathematical entity, an algorithm (or recipe) for maximizing a particular mathematical function with respect to a given collection of data.*“ [Nob06, S. 1565].

Ein wesentlicher Anwendungsbereich der SVMs ist die Klassifikation. Hier besteht der Output der Funktion aus genau einer Klasse aus einer feststehenden Gruppe von Klassen. Ein Beispiel hierfür ist die binäre Klassifikation, bei der einem Objekt genau eines von insgesamt zwei Labeln zugeordnet wird, negativ oder positiv. Jedes Datenobjekt wird dabei durch einen n-Dimensionalen Vektor repräsentiert, welcher entweder der Klasse negativ oder der Klasse positiv zugeordnet ist. Stellt man sich das Ganze grafisch als Punktewolke in einem Diagramm vor, so kann man sagen, dass Objekte mit ähnlichen Eigenschaften sich zu einer Punktewolke beziehungsweise zu einem Cluster zusammenfügen und sich geometrisch durch eine im Diagramm gezogene Linie trennen lassen. Will man neue Objekte klassifizieren, so müsste man entsprechend nach Einordnung in das Diagramm lediglich ablesen, ob das Objekt unterhalb oder oberhalb der entsprechenden Linie liegt. Die sogenannte lineare SVM teilt die Vektoren also mittels dieser separierenden Hyperebene. In Abbildung 3.6 wird deutlich, wie sich die beiden Klassen mittels Hyperebenen linear voneinander separieren lassen. Dabei wird ersichtlich, dass es verschiedene Hyperebenen geben kann, welche die beiden Gruppen korrekt von einander abgrenzen können. Beispiele dafür sind die in Abbildung 3.6 gezeigten Hyperebenen, welche alle die beiden Klassen korrekt voneinander trennen.

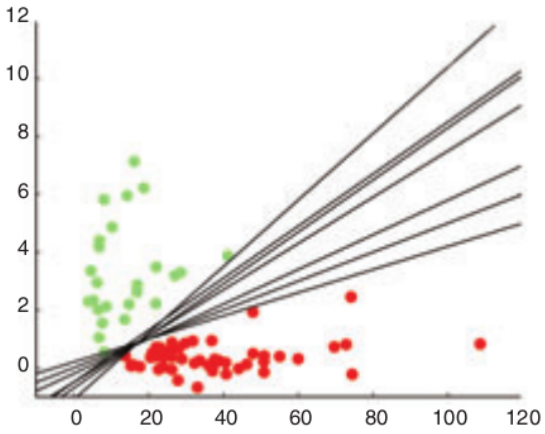

Abbildung 3.6: Beispiel der Linearen Separation von Vektoren durch Hyperebenen [Nob06, S. 1566]

Eine SVM versucht dabei, die Hyperebene so auszuwählen, dass der Abstand zum nächstliegenden Vektor eines Datenobjektes möglichst groß ist. Man spricht dann von der maximum-margin hyperplane [Nob06]. Damit soll erreicht werden, dass neue Datenobjekte zu einem späteren Zeitpunkt besser in eine der Gruppen eingeordnet werden können. Diese Berechnung zählt zu den Optimierungsproblemen beziehungsweise zu den Maximierungsproblemen. Die durch Vektoren repräsentierten Datenobjekte in Abbildung 3.6 stellen die Trainingsdaten der SVM dar. Diese lassen sich mathematisch wie in (3.1) gezeigt darstellen:

$$D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_m, y_m)\} \tag{3.1}$$

Wobei $\vec{x}_i$ den Vektor des zu repräsentierenden Datenobjektes darstellt und y_i die zugehörige Klasse bestimmt. Im Falle der hier erläuterten binären Klassifikation kann y_i somit den Wert -1 oder 1 annehmen.

In Abbildung (3.2) ist zu sehen, wie sich die Funktion F zur Klassifizierung von neuen und somit unbekannten Daten verwenden lässt.

$$F(\vec{x}) \Rightarrow \vec{w} \cdot \vec{x} - b \tag{3.2}$$

Dabei stellt $\vec{w}$ den Normalenvektor zur Gewichtung (weight vector) dar und und b das sogenannte Bias. Diese Werte werden in der Trainingsphase von der SVM erzeugt. In (3.3) wird dargestellt, wie die Zuordnung zur jeweiligen Klasse anhand von $\vec{w}$ und b vorgenommen wird.

$$\begin{aligned} &\vec{w} \cdot \vec{x}_i - b > 0 \text{ wenn } y_i = 1 \text{ , und} \\ &\vec{w} \cdot \vec{x}_i - b < 0 \text{ wenn } y_i = -1 \\ &\text{was zu der folgenden Gleichung umgeformt werden kann:} \\ &y_i(\vec{w} \cdot \vec{x}_i - b) > 0, \forall(\vec{x}_i, y_i) \in D \end{aligned} \tag{3.3}$$

Um nun den Abstand (engl. margin) zu den dichtesten Datenobjekten der Hyperebene zu maximieren wird (3.3) wie folgt umgeformt:

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \forall(\vec{x}_i, y_i) \in D \tag{3.4}$$

Yu (2009) stellt dazu fest, dass wenn D linear separierbar ist oder jeder Vektor in D (3.3) erfüllt, dann existiert auch ein F, das (3.4) erfüllt. Denn wenn es ein $\vec{w}$ und ein b gibt, das (3.3) erfüllt, kann es so umgewandelt werden, dass es auch (3.4) erfüllt [Yu09]. Die Entfernung eines Vektors zu der Hyperebene kann mathematisch wie in (3.5) gezeigt dargestellt werden.

$$\frac{|F(\vec{x}_i)|}{\|\vec{w}\|} \tag{3.5}$$

Und somit wird der margin wie folgt dargestellt:

$$\text{margin} = \frac{1}{\|\vec{w}\|} \tag{3.6}$$

Denn wenn $\vec{x}_i$ die am dichtesten an der Hyperebene liegenden Vektoren sind, wird $F(\vec{x})$ nach (3.4) 1 zurückgeben. In der Abbildung 3.7 wird die Maximierung des margins zusätzlich noch einmal grafisch dargestellt.

Die beschriebene lineare Support Vector Machine ist dabei nicht die einzige Möglichkeit, Objekte im Hyperraum zu klassifizieren.

Ein weiterer Ansatz dafür ist die nonlineare SVM, welche eine nichtlineare Hyperebene konstruiert. Dazu werden die Vektoren wie zuvor in einen hochdimensionalen Raum transformiert und dort wird dann der margin mit Hilfe einer sogenannten Kernel-Funktion erzeugt. Daraus ergibt sich eine flexiblere Methode, um die einzelnen Klassen untereinander abzugrenzen [Ali09]. In

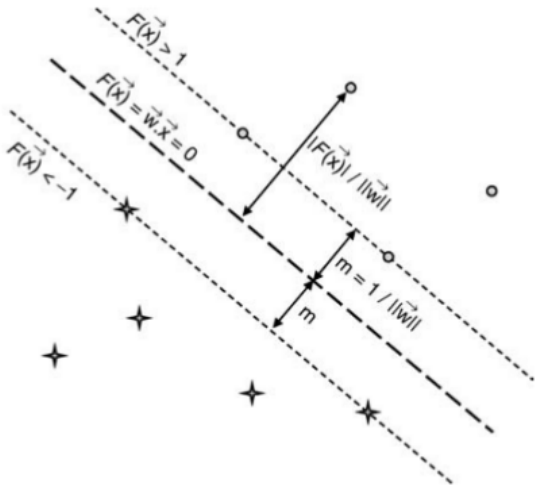

Abbildung 3.7: Die Hyperebene maximiert den Abstand zu den dichtesten Vektoren [Yu09, S. 2891]

der Abbildung 3.8 wird verdeutlicht, wie die Erzeugung einer solchen nichtlinearen Hyperraumebene aussehen kann.

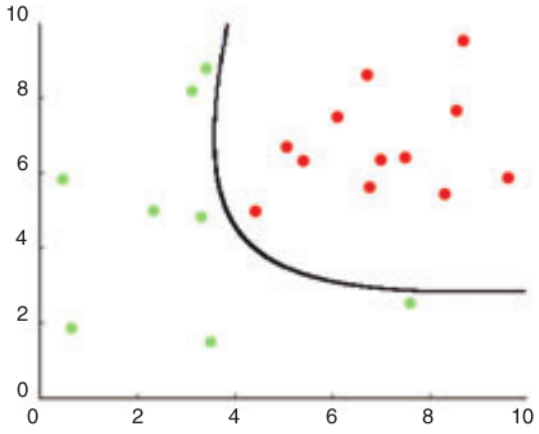

Abbildung 3.8: Beispiel der Separation mittels einer nichtlinearen SVM [Nob06, S. 1566]

Wie zuvor erläutert, bestehen die Trainingsdaten sowohl bei der linearen, als auch bei der nonlinearen SVM meist aus zwei oder mehr Klassen. Diese Art des Anlernens der SVM mittels klassifizierter Daten bezeichnet man als Supervised Learning (siehe 3.3.2). Allerdings gibt es auch Fälle, in denen nur eine einzige Klasse von Daten als Trainingsdaten vorliegt und für die Lernphase verwendet werden kann. Diese Art des Anlernens bezeichnet man als Unsupervised

Learning (siehe 3.3.2). Im Falle des in dieser Arbeit beschriebenen Vorgehens trifft genau diese Situation zu, denn hier soll anhand der Normdaten trainiert werden, um im späteren Verlauf Abweichungen von dieser Norm zu erkennen. Wenn also neue, bisher unbekannte Daten zu sehr von dieser Norm abweichen, sollen sie als außerhalb der gelernten Klasse eingestuft werden. Vlasveld (2013) schreibt hierzu: „*By just providing the normal training data, an algorithm creates a (representational) model of this data. If newly encountered data is too different, according to some measurement, from this model, it is labeled as out-of-class*“ [Vla13].

Ein mögliches Modell für diese Art der Klassifizierung ist die sogenannte One-Class SVM, eine Erweiterung des klassischen SVM Modells, die ebenfalls die Kernel-Funktion nutzt. Wang et al schreiben hierzu: „*The main idea is that the algorithm maps the data into a feature space using an appropriate kernel function, and then attempts to find the hyperplane that separates the mapped vectors from the origin with maximum margin.*“ [WWM04, S. 359]. Dabei ist zu beachten, dass die Trainingsdaten einer One-Class SVM idealerweise frei von Anomalien sind, oder Anomalien zumindest nur in einer kleinen Menge auftreten. Nur auf diese Weise kann die Abgrenzung der gelernten Klasse von den Ausreißern durch den Trainingsvorgang korrekt bestimmt werden. Die One-Class SVM muss die Abgrenzung dabei so weit festlegen, dass möglichst viele der angelernten Daten innerhalb der Klasse liegen. Sie muss dabei allerdings gleichzeitig so eng gefasst sein, dass möglichst viele Ausreißer außerhalb der Klasse liegen. In Abbildung 3.9 ist dabei zu sehen, wie die One-Class SVM den Hyperraum so abgrenzt, dass möglichst viele der Trainingsdaten (weiß) innerhalb der gelernten Begrenzung (rot) liegen. Neue Daten (lila), die nahe bei den gelernten Daten liegen, werden somit auch als ähnlich oder normal eingestuft. Daten, die sich zu sehr von den Trainingsdaten unterscheiden (gelb), werden als außerhalb der gelernten Klasse eingeordnet und gelten somit als Anomalie.

Die erläuterten Beispiele der linearen und nonlinearen SVM sowie der One-Class SVM, beziehen sich jeweils auf Anwendungsfälle der Klassifikation. Neben der Klassifikation werden SVMs auch für Regressionen (Support Vector Regression, SVR) und das sogenannte „preference learning“ (Ranking SVM) genutzt. In der Praxis werden SVMs in verschiedenen Bereichen eingesetzt. So werden sie beispielsweise für die Gesichts- und Handschrifterkennung, die Aufdeckung missbräuchlicher Kreditkartenbewegungen, die Untersuchung von DNA-Profilen oder eben für das Security Monitoring genutzt [Nob06; WWM04].

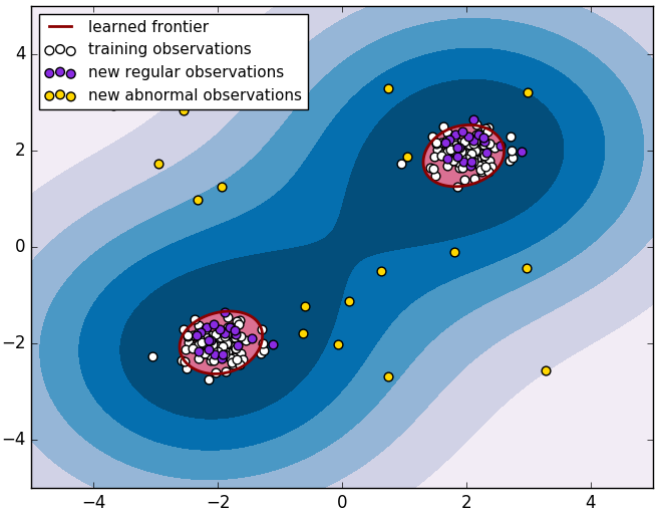

Abbildung 3.9: Beispiel der Anomalieerkennung mittels One-Class SVM [Sci17]

3.3.4 SciPy und Scikit-Learn

Im folgenden Abschnitt werden SciPy und Scikit-Learn vorgestellt. Sie werden für die Implementierung des Machine Learning Teils des in dieser Arbeit vorgestellten Konzeptes eingesetzt.

SciPy ist eine Open-Source Bibliothek für die Programmiersprache Python und stellt eine Reihe von benutzerfreundlichen mathematischen Algorithmen für Signalverarbeitung, Lineare Algebra, Optimierung, Statistik und vieles mehr bereit. SciPy bildet gleichzeitig eine der Kernkomponenten des so genannten SciPy Stacks, welcher eine Sammlung verschiedener Open-Source Software speziell für den Einsatz des wissenschaftlichen Rechnens bereitstellt. Teil dieses Stacks sind unter anderem NumPy (Handhabung von Vektoren, Matrizen, etc.), matplotlib (Generierung mathematischer Darstellungen) und nose (Testing Bibliothek).

Scikit-Learn wiederum ist ein Python Toolkit welches auf der SciPy Softwareumgebung basiert und unabhängig davon entwickelt wird. Der Name Scikit ist die Kurzform für SciPy Toolkit, wobei Scikit-Learn eines von vielen Toolkits für verschiedene wissenschaftliche Schwerpunkte darstellt. Der Fokus von Scikit-Learn liegt auf der Bereitstellung von einfach zu verwendenden Machine Learning Methoden wie dem Supervised Learning und dem Unsupervised Learning mittels unterschiedlicher Algorithmen. Pedregosa (2011) beschreibt das Toolkit als: „*Scikit-learn harnesses this rich environment to provide state-of-the-art implementations of many well known machine learning algorithms, while maintaining an easy-to-use interface tightly inte-*

grated with the Python language. This answers the growing need for statistical data analysis by non-specialists in the software and web industries, as well as in fields outside of computer-science, such as biology or physics.“ [PVG+11, S. 2826]. Teil des Scikit-Learn Toolkits ist auch die in dieser Arbeit verwendete One-Class SVM zur Anomalieerkennung.

3.3.5 Knowledge Discovery Process

Die Menge an digitalen Daten nimmt stetig und rapide zu, was dazu führt, dass es immer bedeutsamer wird, Theorien beziehungsweise Techniken zu entwickeln, die es ermöglichen, aus der enormen Menge an Daten „nützliche“ Informationen (knowledge) zu extrahieren [FPSS96a]. Der Vorgang der Wissensgewinnung aus Rohdaten, wird als **Knowledge Discovery in Databases Process** (KDD) bezeichnet. Fayyad et al (1996) beschreiben diesen Prozess als: „*[…] the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data. At the core of the process is the application of specific data-mining methods for pattern discovery and extraction.*“[FPSS96a, S. 37 & 40,41]. Der Prozess ist dabei interaktiv und iterativ und beinhaltet verschiedene Schritte, in denen jeweils Entscheidungen des Nutzers einfließen, und die in Abbildung 3.10 überblicksartig dargestellt sind.

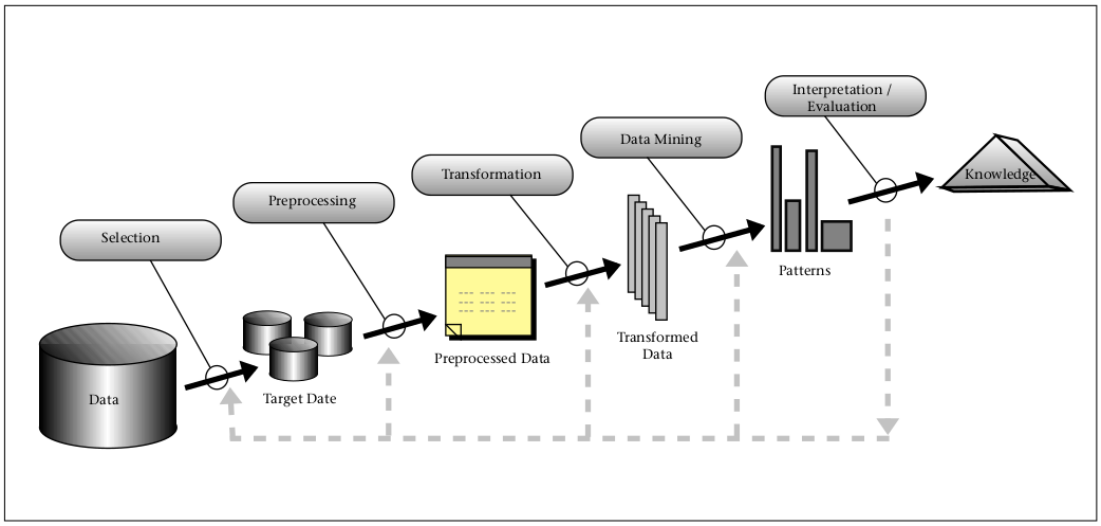

Abbildung 3.10: Knowledge Discovery Process [FPSS96a, S. 41]

Am Anfang des KDD-Prozesses muss das Anwendungsfeld festgelegt werden beziehungsweise bestimmt werden, welches Ziel (z.B. Identifikation von Malware) durch den Prozess erreicht

werden soll und welche Daten hierfür relevant sind. Sind diese Aspekte festgelegt worden, wird durch die Selection die sogenannte Target Data erzeugt. Im Schritt des Preprocessing werden die Daten für die weiteren Schritte vorbereitet, indem beispielsweise festgelegt wird, wie mit unvollständigen Daten umgegangen werden soll. Die so vorbereiteten Daten werden in der Transformation für die Weiterverarbeitung umgewandelt. Hier werden unter anderem Eigenschaften festgelegt, die die Daten sinnvoll repräsentieren. Außerdem werden Methoden der Dimensions-Reduktion beziehungsweise der Transformation genutzt, um die tatsächliche Anzahl der berücksichtigten Variablen weiter zu reduzieren beziehungsweise unveränderliche Repräsentationen der Daten zu ermitteln. Im nächsten Schritt wird der zu nutzende Data-Mining Algorithmus festgelegt. Hierfür muss festgelegt werden, welches Ziel (z.B. Klassifikation oder Regression) durch den Algorithmus erreicht werden soll. Dies beinhaltet auch, dass eine Methode ausgewählt werden muss, wie in den Daten nach Mustern gesucht werden soll und dass diese Methode sowohl für das gewünschte Ziel sinnvoll ist, als auch vom Nutzer verstanden wird. Im finalen Schritt der Interpretation werden die durch das Data-Mining gefundenen Muster gedeutet beziehungsweise bewertet und gegebenenfalls visualisiert. Hierfür können redundante oder irrelevante Muster aussortiert und die relevanten Muster in nutzerfreundlicher Form aufbereitet werden.

Zusammenfassend kann gesagt werden, dass der KDD-Prozess eine Vielzahl an Schritten enthält, die es ermöglichen, aus einem Datensatz die für eine Zielsetzung relevanten Informationen herauszufiltern und aufzubereiten. Der Prozess läuft dabei nicht rein linear ab, sondern es kann Schlaufen zwischen den einzelnen Schritten geben. Das durch den KDD-Prozess gewonnene Wissen kann dann beispielsweise dokumentiert, an den Nutzer oder andere Sakeholder berichtet, mit vorherigem Wissen abgeglichen und zum Einleiten weiterer Handlungsschritte genutzt werden.

4 Konzept

Wie bereits erläutert, soll in der vorliegenden Arbeit ein Konzept für eine Softwarearchitektur vorgestellt werden, die es ermöglicht, mittels eines intelligenten Systems Security-Monitoring für IoT Endgeräte zu betreiben. Der grundlegende Gedanke hinter diesem Konzept ist es, eine Abweichung des Verhaltens von IoT Endgeräten nach einer Malware Infektion im Datenverkehr zu erkennen und entsprechende Warnungen auszugeben. Das Konzept umfasst Empfehlungen für die Erfassung des Datenverkehrs auf Netzwerkebene, die Speicherung der Daten auf einem Suchserver, die Auswertung der Daten mittels Machine Learning sowie die Visualisierung der Ergebnisse für den Benutzer in einer Webanwendung. Diese Aspekte, die auch in Abbildung 4.1 dargestellt sind, werden nachfolgend näher erläutert. Der Konzeptteil zur Datenverarbeitung und Visualisierung wird dabei anhand des KDD-Prozesses aufgebaut.

4.1 Datenerfassung

Als zu erfassende Daten wird im vorliegenden Konzept der Datenverkehr eines lokalen Netzwerks und im Speziellen der Datenverkehr der im Netzwerk befindlichen IoT Endgeräte gewählt. Der Datenverkehr besteht aus einzelnen Datenpaketen aus der Familie des Internet Protocols (IP). Sie beinhalten unter anderem die für dieses Konzept betrachteten Daten über den Sender und Empfänger des Pakets, den Quell- und Zielport sowie die Größe des jeweiligen Pakets, wie in Abbildung 4.2 dargestellt.

Um den Datenverkehr des gesamten Netzwerks erfassen zu können, muss dieser auf eine Maschine im Netzwerk gespiegelt werden, wofür es verschiedene Möglichkeiten gibt. Ein in diesem Konzept favorisierter Ansatz dafür ist das sogenannte Port Mirroring, wobei ein Router so eingerichtet wird, dass er eine Kopie sämtlicher Datenpakete an eine bestimmte Zieladresse eines Systems weiterleitet. Werden die Datenpakete nun auf eine Maschine gespiegelt, können sie dort für die Weiterverarbeitung in Empfang genommen werden. In diesem Konzept wird

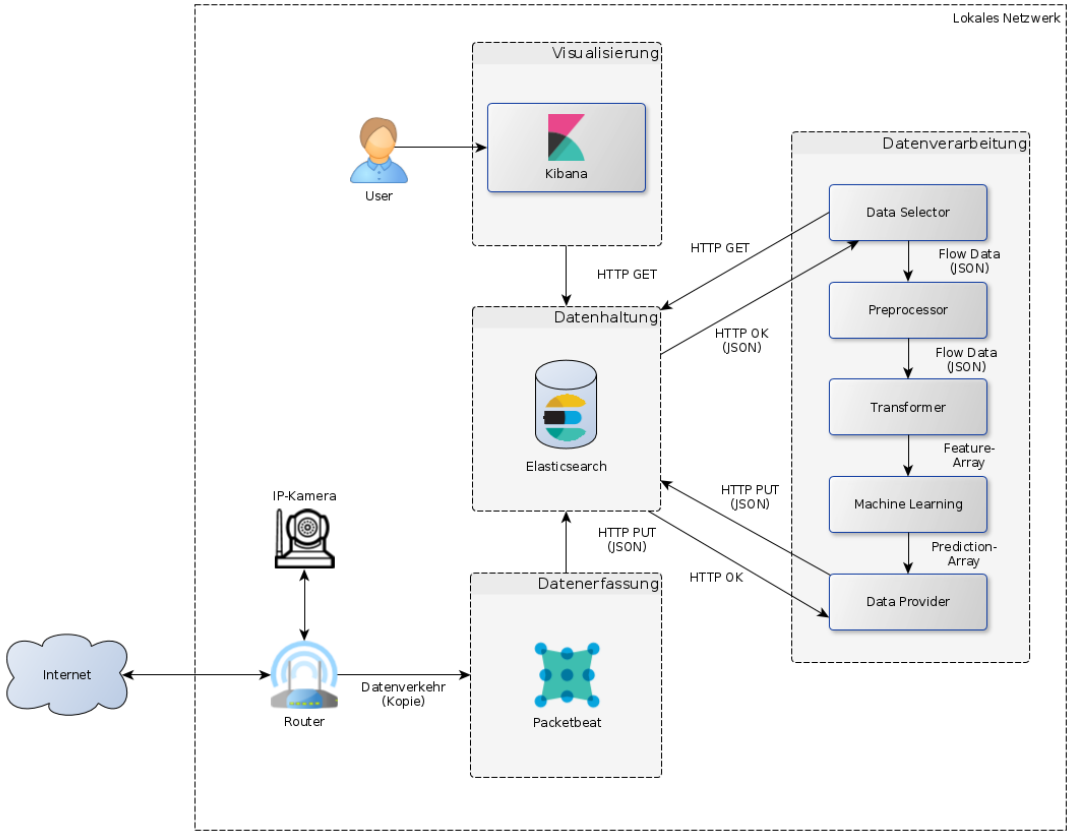

Abbildung 4.1: Softwarearchitektur Konzept[1]

dafür das Programm Packetbeat der Firma Elasticsearch BV genutzt, welches unter der Apache 2.0 Lizenz zur Verfügung gestellt wird.

Packetbeat bietet dabei nicht nur die Möglichkeit, die ankommenden Daten in Empfang zu nehmen, sondern ermöglicht es auch, einige Application Layer Protokolle zu dekodieren. Darunter sind HTTP, MySQL und Redis. Die Dekodierung ermöglicht einen Einblick in die Nutzdaten der Pakete, welche für Statistiken und Optimierungen verwendet werden können. So können beipielsweise häufig verwendete Datenbankanfragen oder eine Netzwerk Topologie dargestellt werden. Im vorliegenden Konzept wird Packetbeat für die Datenerfassung und nicht für die Dekodierung genutzt. Auch ohne die Dekodierung gewinnt Packetbeat einige Metainformationen aus den Daten indem beispielsweise Netzwerkanfragen mit den dazugehörigen

[1] IP Camera Icon by Arthur Shlain from the Noun Project — Creative Commons 3.0

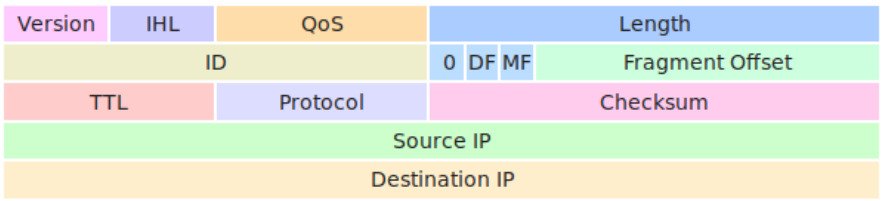

Abbildung 4.2: Headerinformationen eines IPv4-Pakets [Wik17]

Antworten korreliert werden. Daraus können dann Informationen wie etwa Antwortzeiten, der Status eines Systems und sogenannte Flowdaten gewonnen werden. Ein Flow ist eine Zusammenfassung der Übertragung mehrerer Datenpakete zwischen zwei Kommunikationspartnern auf dem selben Kanal (Protokoll und Port) über einen definierten Zeitraum. Der Flow enthält dann hauptsächlich die folgenden Informationen: Die Adresse der beiden Kommunikationspartner (Source und Destination IP), das verwendete Protokoll (UDP, TCP etc.), den Quell- und Ziel-Port, die übertragene Datenmenge innerhalb des Flows in Bytes sowie die Anzahl einzelner übertragener Datenpakete des Flows. In dieser Arbeit liegt der Schwerpunkt auf der Verarbeitung der genannten Flowdaten, da diese bereits korrelierte Informationen über die Kommunikationspartner sowie die ausgetauschten Datenmengen im lokalen Netzwerk beinhalten und somit eine ideale Grundlage für Weiterverarbeitung bieten. Als letzten Schritt ermöglicht Packetbeat die Umwandlung der Daten in das JSON Format, welches von vielen anderen Anwendungen für die Weiterverarbeitung eingelesen werden kann. Die Abbildung 4.3 stellt die soeben vorgestellten Verarbeitungsschritte von Packetbeat grafisch dar.

Abbildung 4.3: Packetbeat Verarbeitungsschritte[2]

Diese Art der Datenerfassung wird dann genutzt, um das *Normverhalten* eines IoT Geräts zu erfassen. Also den Datenverkehr, den das Gerät bei üblicher Verwendung produziert ohne von einer Malware oder ähnlichem infiziert zu sein. Diese Daten werden im späteren Verlauf zum Trainieren des intelligenten Systems verwendet.

[2]Bildquelle: https://www.elastic.co/products/beats/packetbeat, Zugriff am 17.03.2017

4.2 Datenhaltung

Die Datenhaltung dieses Konzepts beschreibt, wie die gewonnen Daten (siehe Abschnitt 4.1) zur späteren Verwendung geordnet abgelegt werden können. Dafür stehen grundsätzlich diverse Möglichkeiten und verschiedene Datenbanksysteme zur Verfügung. Die traditionellen relationalen Datenbankmanagementsysteme (RDBMS), wie beispielsweise die SQL Datenbank, sind typischerweise tabellenbasiert und benötigen im Vorfeld definierte Schemata, welche den Tabellenaufbau beschreiben [Cat11]. Diese Tabellen können dann in Relation zueinander gebracht werden. Sogenannte NoSQL Datenbanken hingegen weisen andere Merkmale auf und können in unterschiedliche Datenmodelle unterteilt werden. Dies sind zum Beispiel die Key-Value Datenbanken, bei dem ein gespeicherter Wert immer einem eindeutigen Schlüssel zugeordnet wird. Zudem gibt es Spaltenorientierte Datenbanken, wobei Tabellen als Datenmodell eingesetzt werden, welche allerdings keine Assoziationen untereinander zulassen. Außerdem werden noch Dokumentenorientierte Datebanken unterschieden, welche Ähnlichkeit zur Key-Value Datenbank haben, sich aber von diesen dahingehend unterscheiden, dass als Wert immer ein Dokument gespeichert wird. Das Dokument liegt dabei oft im XML oder JSON Format vor [HELD11]. Auch weitere Modelle wie beispielsweise für die Speicherung von stark vernetzten Informationen mittels einer Graphdatenbank stehen zur Verfügung.

Im vorliegenden Konzept wird für die Datenhaltung eine NoSQL Datenbank verwendet, der Suchserver Elasticsearch. Elasticsearch wurde auf Basis der Apache Lucene Bibliothek zur Volltextsuche entwickelt und kann als eine dokumentenbasierte NoSQL Datenbank verwendet werden. Die Software steht unter der Open-Source Apache 2.0 Lizenz und ist frei verfügbar. Da es sich hierbei um ein dokumentenbasiertes System handelt, werden in diesem Fall immer JSON Dokumente in der Datenbank abgelegt. Die Besonderheit eines Suchservers ist, dass ein Index erstellt wird, welcher beim Anlegen neuer Datensätze kontinuierlich gepflegt wird. Der Index enthält dabei Informationen über die gepeicherten Daten und ermöglicht eine umfangreiche Suchfunktion. Elasticsearch indiziert also zu erfassende Dokumente direkt bei Eingang und speichert sie zusammen mit einigen Metadaten als Dokument im JSON Format ab. Dokumente ähnlicher Struktur werden dabei zu Typen zusammengefasst und mehrere Typen bilden in Elasticsearch dann einen Index. Dokumente können dann über eine REST (Representational State Transfer) API (Application Programming Interface) erstellt und abgerufen werden. Für Abfragen steht eine Abfragesprache in Form der sogenannten Query DSL (Domain Specific Language) zur Verfügung.

Das im Abschnitt 4.1 beschriebene Verfahren zur Datenerfassung mittels der Packetbeat Anwendung, kann über die REST API von Elasticsearch angebunden werden. Da Packetbeat die erfassten Flowdaten bereits als JSON Dokument ausgeben kann, können diese ohne weitere Vorverarbeitung direkt in den Suchserver gegeben werden. Wurden die Daten in Elasticsearch abgelegt, kann über die Query DSL darauf zugegriffen werden. Eine einfache Suchanfrage um ein gespeichertes JSON Dokument des Typs *flow* aus dem Index *packetbeat* abzurufen, könnte dann wie folgt aussehen:

```
$ curl -X GET 'http://localhost:9200/packetbeat*/_search' -d '
{
  "query": {
    "term": {
      "type": "flow"
    }
  },
  "size": 1
}'
```

Listing 4.1: Elasticsearch Flowdaten Query

Zusammenfassend lässt sich sagen, dass die Datenhaltung im vorliegenden Konzept dokumentenbasiert mittels Elasticsearch erfolgt und die Daten im JSON Format gespeichert werden.

4.3 Datenverarbeitung

Die Aufgabe der Datenverarbeitung als Teil dieses Konzepts besteht wie einleitend erwähnt in der Erkennung von Auffälligkeiten des Kommunikationsverhaltens beispielsweise nach einer Malware Infektion von im lokalen Netzwerk befindlichen IoT Endgeräten. Die Grundlage für diesen Vorgang bildet der von den im Netzwerk befindlichen Geräten erzeugte Datenverkehr. Dieser wird wie in Abschnitt 4.1 beschrieben erfasst und wie im vorangehenden Abschnitt 4.2 dargestellt gespeichert. Es wird nun in diesem Abschnitt dargestellt, wie künstliches Wissen über den Zustand der IoT Geräte mittels eines intelligenten Systems aus den vorliegenden Daten gewonnen werden kann. Dieser Vorgang der Wissensgewinnung aus den Rohdaten basiert als Teil dieses Konzeptes auf dem **Knowledge Discovery Process**. Wie in Abschnitt 3.3.5 beschrieben, besteht dieser aus einzelnen Schritten, welche nachfolgend in einer an das Konzept angepassten Form vorgestellt werden.

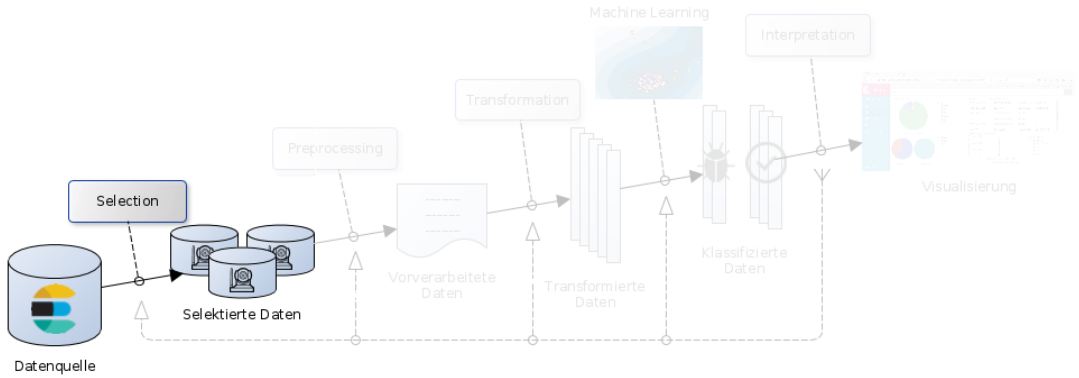

Abbildung 4.4: Knowledge Discovery Process — Selection [nach FPSS96a, S. 41]

Der erste Schritt besteht darin, eine Vorauswahl der Rohdaten zu treffen. Die sogenannte **Selection** (Abbildung 4.4). Im Falle dieses Konzepts werden hier zunächst die Daten des Typs *flow* aus dem Bestand des gesamten Datenverkehrs ausgewählt, da diese wie in Abschnitt 4.1 erläutert bereits korrelierte Informationen über die Kommunikation der Geräte enthalten. Diese Daten werden weiterhin so gefiltert, dass nur noch die Flowdaten enthalten sind, bei denen einer der beiden Kommunikationspartner (Sender oder Empfänger) ein IoT Endgerät ist. So wird sichergestellt, dass nur genau der Datenverkehr analysiert wird, bei dem auch ein IoT Gerät beteiligt war. Der Datenverkehr anderer Geräte im lokalen Netzwerk ist nicht Teil dieser Arbeit. Eine weitere Eingrenzung des analysierten Datenverkehrs besteht in der Betrachtung eines begrenzten Zeitraums. Hierbei kann beispielsweise der Datenverkehr einer Woche, eines Tages oder einiger Minuten für die Verarbeitung ausgewählt werden. Dabei kommt es darauf an, in welchen zeitlichen Abständen Auswertungen durchgeführt werden sollen, was je nach Anwendungsfall variieren kann.

Im zweiten Schritt werden die zuvor ausgewählten Daten für den Prozess des Machine Learnings vorbereitet. Die Aufgabe des sogenannten **Preprocessors** (Abbildung 4.5) ist es, die Daten vorab zu säubern und Strategien für den Umgang mit fehlenden Informationen in den Datensätzen umzusetzen [FPSS96b]. In diesem Schritt werden innerhalb dieses Konzeptes unvollständige Datensätze aus der Verarbeitung ausgenommen, um die Ergebnisse nicht zu verfälschen.

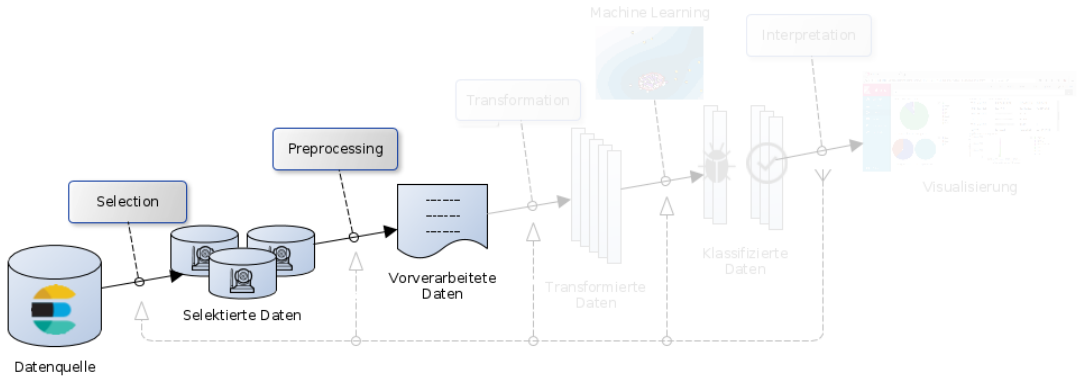

Abbildung 4.5: Knowledge Discovery Process — Preprocessing [nach FPSS96a, S. 41]

Bis zu diesem Zeitpunkt liegen die Daten noch im JSON Format vor. Im nächsten Schritt werden die Daten in ein für das Machine Learning lesbares Format umgewandelt. Dieser Schritt wird als **Transformation** (Abbildung 4.6) bezeichnet. Da der für die Anomalieerkennung genutzte One-Class SVM Algorithmus ein numerisches Array voraussetzt, werden die Daten entsprechend transformiert. Dieser Schritt beinhaltet außerdem die Auswahl der sogenannten Features, also der Datenfelder, welche zum Lernen und für den Voraussage-Prozess verwendet werden sollen [FPSS96b]. Die vorliegenden Flowdaten werden also aus dem JSON Format ausgelesen, in numerische Werte umgewandelt und anschließend in einem Array abgelegt. Die Abbildung 4.7 zeigt die Werte der Flowdatenfelder, die als Features verwendet werden.

Da die IP-Adressen vier durch Punkte von einander getrennte Zahlenblöcke besitzen (Beispiel: 127.0.0.1), liegen sie zunächst noch als Zeichenkette (String) vor. Strings stellen für eine SVM eine ungültige Eingabe dar, daher werden sie durch eine Aufteilung in vier einzelne numerische Features pro IP-Adresse umgewandelt (pro Zahlenblock ein Feature). In diesem Zug werden die Features auch normalisiert, also in einen Zahlenbereich zwischen 0 und 1 umgerechnet. Dies geschieht, indem der Wert des Zahlenblocks durch den Maximalwert geteilt wird. Die Zahl eines Blocks einer IPv4 Adresse liegt im Bereich zwischen 0 und 255. Für die Normalisierung der IP-Adressen werden die einzelnen Zahlenblöcke somit durch 255 geteilt.

Die in Abbildung 4.7 aufgeführten Features werden im nächsten Schritt an die **Machine Learning** (Abbildung 4.8) Komponente übergeben. Dieses Feature-Array wird dabei für den Lernprozess und für die darauf folgende Berechnung der Voraussage verwendet. Mittels des

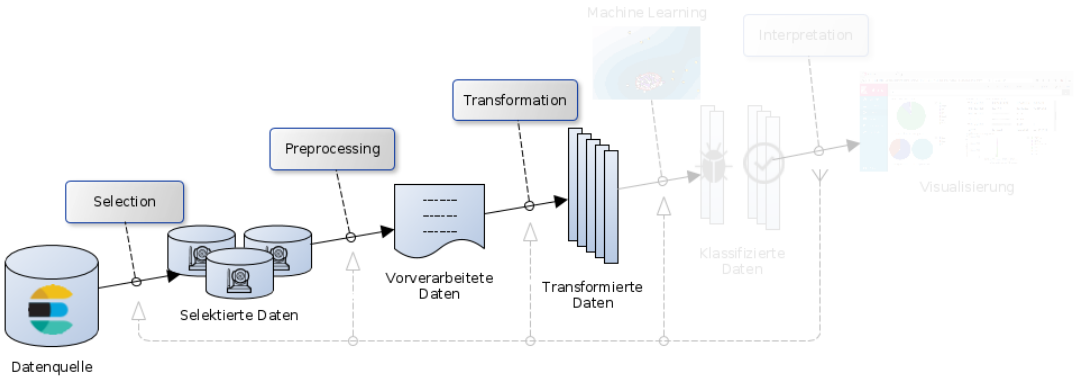

Abbildung 4.6: Knowledge Discovery Process — Transformation [nach FPSS96a, S. 41]

Feature	Beschreibung
Quell-IP-Adresse	IPv4 Adresse des Senders
Ziel-IP-Adresse	IPv4 Adresse des Empfängers
Quell-Port	verwendeter Port des Senders
Ziel-Port	verwendeter Port des Empfängers
Anzahl Quell-Pakete	Anzahl der vom Sender übertragenen Pakete des Flows
Anzahl Ziel-Pakete	Anzahl der vom Empfänger übertragenen Pakete des Flows
Quell Paketgröße	Größe der übertragenen Pakete in Bytes
Ziel Paketgröße	Größe der übertragenen Pakete in Bytes

Abbildung 4.7: Liste der verwendeten Features aus den Flowdaten

Lernprozesses wird das *Normverhalten* eines Systems erlernt, welches dann als Grundlage für zukünftige Voraussagen verwendet wird, indem es mit dem aktuellen Profil verglichen wird.

Wie bereits in Abschnitt 3.3.4 berwähnt, wird in diesem Konzept der sogenannte One-Class SVM aus dem Bereich der Anomalieerkennung mittels Machine Learning aus dem Scikit-Learn Toolkit verwendet. Die One-Class SVM ermöglicht es, ohne eine vorangehende Klassifikation der Daten in der Lernphase, spätere Voraussagen über Abweichungen von der Norm treffen zu können. Damit fällt sie in die Klasse der Machine Learning Methoden des Unsupervised Learnings. Die Tatsache, dass bei der One-Class SVm nur Normaldaten für die Lernphase benötigt werden, ist gleichzeitig auch der entscheidende Grund dafür, dass für dieses Konzept die One-Class SVM gegenüber anderen Machine Learning Verfahren bevorzugt wurde. Dass in der Lernphase nur eine Klasse von Daten benötigt wird ist sinnvoll, da es zum Teil kaum

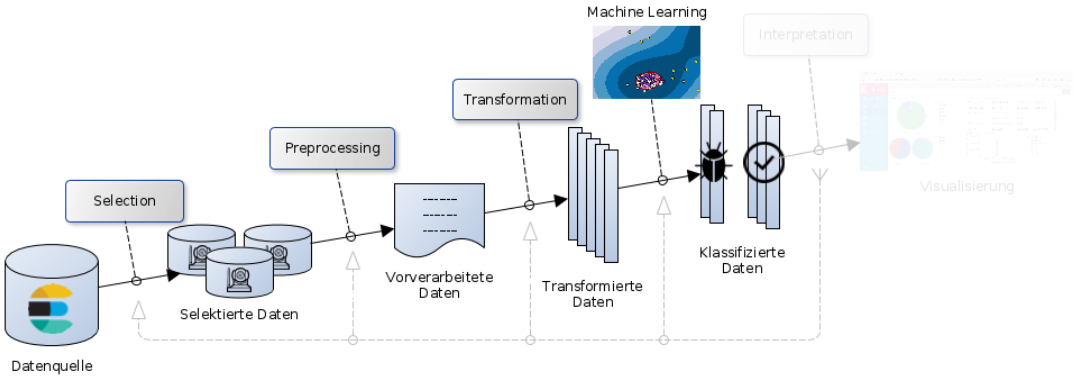

Abbildung 4.8: Knowledge Discovery Process — Machine Learning [nach FPSS96a, S. 41]

oder gar nicht möglich ist, Schaddaten für den Lernprozess zu erhalten. Außerdem wäre bei der Nutzung von speziellen Schaddaten in der Lernphase die Gefahr vorhanden, dass nur die Erkennung dieses speziellen Schadprogramms trainiert wird und andere Schadprogramme später nicht erkannt werden.

Nach Abschluss der Lernphase werden Voraussagen über die eingegebenen Flowdaten, die sogenannten Observations (Beobachtungen) getroffen. Eine Observation entspricht dabei in diesem Konzept einem Flow in Form eines Feature-Arrays. Die One-Class SVM wertet jede Observation einzeln aus und vergleicht diese mit den in der Lernphase erfassten Normdaten. Als Ergebnis liefert die One-Class SVM entweder eine 1, wenn die Observation innerhalb der Norm liegt. Oder aber eine -1, wenn die Observation als Anomalie erkannt wurde und somit eine Abweichung von der Norm darstellt.

Die ermittelten Ergebnisse werden im Anschluss an den Data Provider übergeben, welcher sie mit Metadaten zu den jeweiligen Observations anreichert und in das JSON Format umwandelt. Daraufhin wird eine Verbindung zur Datenhaltung aufgebaut und die Ergebnisse werden dort in einem eigenen Index *results* ablegt. Im weiteren Verlauf kann die Visualisierungsschicht auf die Ergebnisse zugreifen und diese grafisch aufbereiten.

4.4 Visualisierung

Die Visualisierung dient dazu, die Ergebnisse der Datenverarbeitung für den Nutzer aufzubereiten und darzustellen. Sie stellt zugleich den letzten Schnitt des Knowledge Discovery Process dar (Abbildung 4.9). Hierfür wird in diesem Konzept die Webanwendung Kibana der Firma Elasticsearch BV eingesetzt. Diese bietet bereits eine nahtlose Anbindung an den Elasticsearch Suchserver, welcher innerhalb dieses Konzeptes für die Datenhaltung verwendet wird. Mit Kibana lassen sich mittels der Elasticsearch Query DSL Anfragen an den Suchserver stellen, wobei die Ergebnisse direkt im Browser eingesehen werden können. Zusätzlich zur Einsicht in einzelne Datensätze lassen sich die Suchergebnisse auch zusammenfassen und grafisch darstellen. Dafür stehen verschiedene Visualisierungswerkzeuge zur Verfügung, wie beispielsweise Tabellen, Bereichs-, Torten- und Liniendiagramme sowie Karten. Die Visualisierungswerkzeuge lassen sich mit Abfragen und Aggregationen anpassen und können in sogenannten Dashboards gruppiert und gespeichert werden. Die Abbildung 4.10 zeigt ein Beispiel eines benutzerdefinierten Dashboards in Bezug auf die vorhandenen Flowdaten.

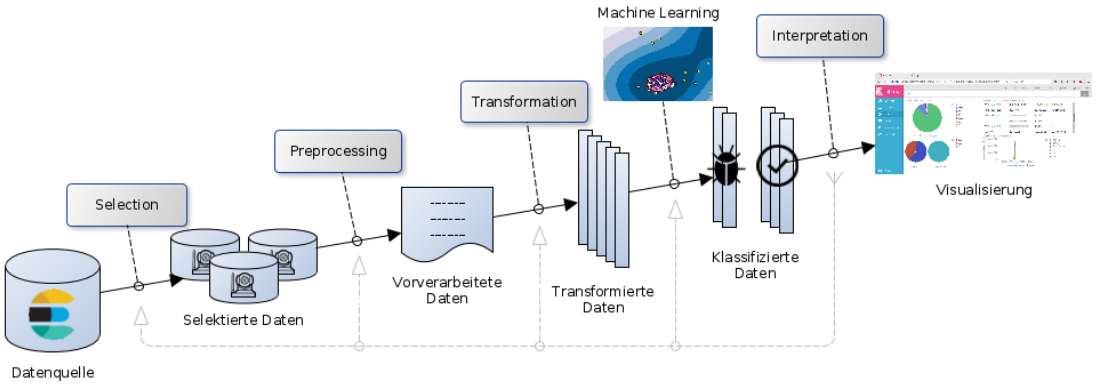

Abbildung 4.9: Knowledge Discovery Process — Interpretation [nach FPSS96a, S. 41]

Für komplexere Aufgaben ermöglicht Kibana die Entwicklung von Plugins. Diese Plugins werden in Javascript geschrieben und bieten die Möglichkeit, Kibana zu einem Security-Monitoring Frontend auszubauen, was hier beispielhaft dargestellt werden soll. Das Plugin soll zunächst die Möglichkeit bieten, eine Liste zu verwalten, in der die zu überwachenden IoT Endgeräte mittels IP-Adresse oder Hostname gespeichert werden. Zum Führen dieser Liste gibt es zum einen die Möglichkeit, eine Blacklist zu verwenden, bei der alle Geräte überwacht werden, außer denen, die sich auf der Liste befinden. Zum anderen kann die Liste auch als

Whitelist geführt werden, wobei nur Geräte überwacht werden, die explizit auf der Liste stehen. In dem hier dargestellten Konzept soll eine Whitelist verwendet werden. Diese Liste steht dann der Machine Learning Komponente zur Verfügung und kann per Query abgefragt und für die Filterung des Datenverkehrs im Data Selector verwendet werden. So ist eine gezielte Auswahl einzelner Geräte im lokalen Netzwerk möglich. Außerdem soll das Plugin die Möglichkeit bieten, die als Anomalie gekennzeichneten Flowdaten übersichtlich darzustellen. Hierzu bietet sich beispielsweise die Verwendung eines Liniendiagramms an, welches die Anzahl der Anomalien pro Gerät auf einer Zeitachse zeigt. Mithilfe der *Alerting*[3] X-Pack Erweiterung für Elasticsearch ist es zudem möglich, Warnungen bei Überschreitung von Grenzwerten über verschiedene Wege zu verschicken, darunter auch per Mail. Integriert man diese Erweiterung in das hier beschriebene Plugin, ist es für den Nutzer möglich, automatisch über aktuelle Auffälligkeiten in den Verkehrsdaten der IoT Geräte informiert zu werden.

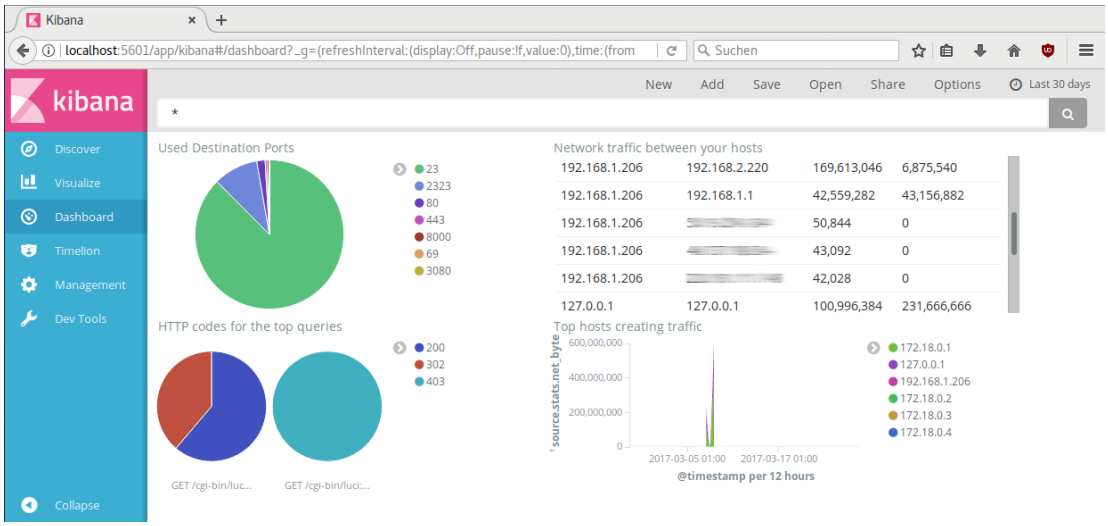

Abbildung 4.10: Beispiel eines Kibana Dashboards

In den vorangegangen Abschnitten wurde ein Konzept für eine Softwarearchitektur vorgestellt, die es ermöglicht, mittels eines intelligenten Systems Security-Monitoring für IoT Endgeräte zu betreiben. Neben Empfehlungen für die Datenerfassung,die Datenhaltung sowie die Datenverarbeitung, wurden auch Visualisierungsmöglichkeiten dargestellt. Beachtet werden muss an dieser Stelle, dass für die Umsetzung dieses Konzeptes zwischen der Lernphase und der Anwendungsphase unterschieden werden muss. Nach der Initiierung der Lernphase durch den Nutzer, wird der Datenverkehr des Netzwerks erfasst, gespeichert und verarbeitet. In

[3] https://www.elastic.co/products/x-pack/alerting

der Lernphase wird als Teil des Verarbeitungsprozesses die One-Class SVM mit diesen Daten trainiert. Der Nutzer legt das Ende der Lernphase über das Kibana Plugin fest. Nach Abschluss der Lernphase werden die Netzwerkdaten weiterhin erfasst, gespeichert und verarbeitet, mit dem Unterschied, dass die One-Class SVM die eingegangenen Daten nicht mehr als Trainingsdaten, sondern für die Voraussage beziehungsweise Einordnung in Schaddaten vs. keine Schaddaten verwendet. Die gewonnen Ergebnisse können wiederum visualisiert und dem Nutzer bereitgestellt werden.

Zusammengefasst stellen die erläuterten Aspekte ein Gesamtkonzept einer Softwarearchitektur zum Security-Monitoring für IoT Endgeräte mittels eines intelleginten Systems dar, wie es in die Praxis übertragen werden kann.

5 Implementierung

Auf Basis des in Abschnitt 4 dargestellten Konzeptes, soll in der vorliegenden Arbeit anhand eines Prototypen gezeigt werden, wie mittels eines intelligenten Systems Security-Monitoring für IoT Endgeräte betrieben werden kann. Der Fokus des Implementierungsteils liegt dabei auf der Datenerfassung, Datenhaltung und Datenauswertung.

Nachfolgend werden zunächst der Testaufbau der Hardware und die damit verbundene Konfiguration dargestellt. Darüber hinaus wird beschrieben, wie Machine Learning mittels Scikit-Learn in der Implementierung des Prototypen genutzt wurde.

5.1 Testaufbau und Konfiguration

Um das IoT Security-Monitoring mittels Machine Learning unter möglichst realen Bedingungen testen zu können, kam statt einer reinen Software Simulation ein physikalisches lokales Netzwerk bestehend aus einem Router (D-Link DIR-615), einem Monitoring Server (Desktop mit Betriebssystem Fedora 25), einem CNC (Command & Control) Server (Laptop mit Betriebsystem Fedora 25), einer IoT IP-Kamera (LogiLink Wireless IP Camera WC0030A) und einem Smartphone (Android Betriebssystem Version 5.1.1) zur Anwendung. Das Smartphone diente dabei der Steuerung der Kamera durch eine App des Herstellers.

Der Fokus der vorliegenden Arbeit liegt auf der Auswertung des Datenverkehrs der verwendeten IoT Endgeräte, in diesem Fall der genannten IP-Kamera. Der Datenverkehr läuft dabei über den Router oder einen Access Point (AP) im lokalen Netzwerk und wird von dort als Kopie per sogenanntem Port Mirroring an den Monitoring Server zur Auswertung weitergeleitet. Theoretisch könnte die Auswertung der Daten auch direkt auf dem Router oder AP durchgeführt werden, allerdings stehen diesen Geräten oftmals nur sehr beschränkte Ressourcen zur Verfügung. Außerdem würde eine Überlastung dieser Knotenpunkte im lokalen Netzwerk zur Beeinträchtigung des gesamten Datenverkehrs führen, was für eine Monitoring Anwendung

nicht wünschenswert wäre. Der verwendete Router lief mit der Open Source Firmware OpenWrt in der Version 15.05.1, welche für eine Vielzahl an Routermodellen zur Verfügung steht. Die Abbildung 5.1 zeigt eine exemplarische Darstellung einer Netzwerkarchitektur, in der ein Monitoring Server den Datenverkehr auswerten kann.

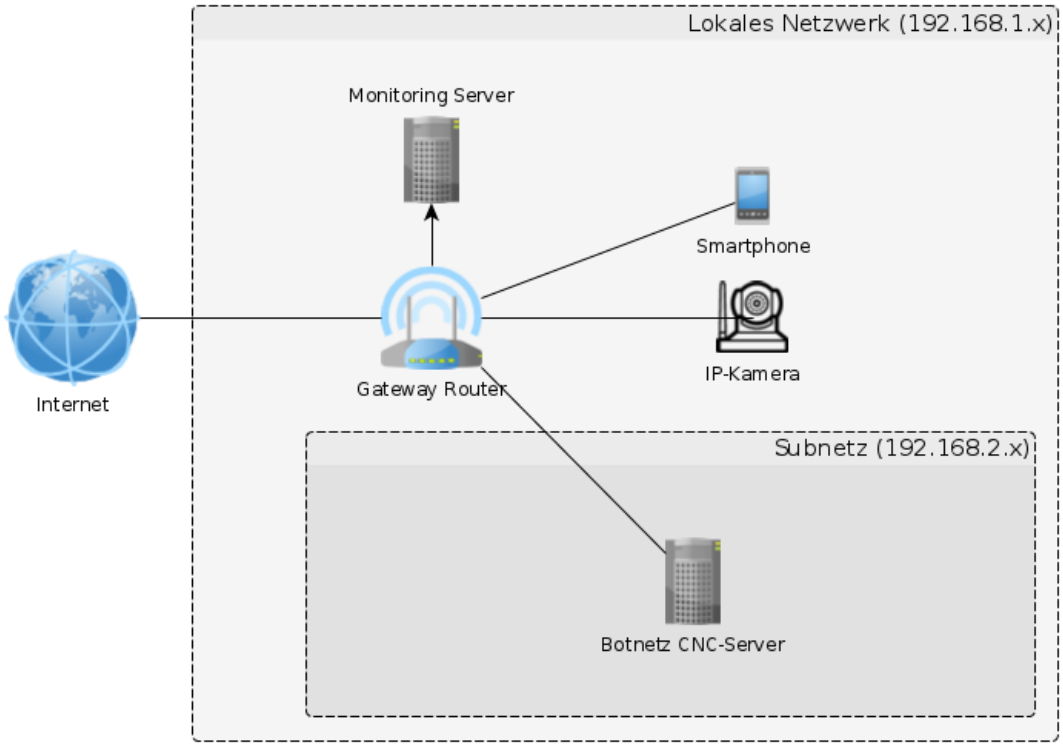

Abbildung 5.1: Netzwerkarchitektur des Testaufbaus[1]

5.1.1 Router Konfiguration

Bei dem in diesem Testaufbau verwendeten Router handelt es sich um das Modell D-Link DIR-615 auf dem die Open-Source Router Firmware OpenWrt installiert wurde. Der Router muss als Teil des Testaufbaus so konfiguriert werden, dass dieser den Datenverkehr des lokalen Netzwerkes als Kopie auf den Monitoring Server weiterleitet. Dies ist beispielsweise mittels des sogenannten Port Mirrorings möglich, so wie es im Rahmen dieses Testaufbaus verwendet

[1] IP Camera Icon by Arthur Shlain from the Noun Project — Creative Commons 3.0

und nachfolgend erläutert wird. Um das Port Mirroring in OpenWrt auf dem Router einrichten zu können, wurde das iptables Modul *iptables-mod-tee* benötigt. Dieses Modul kann mittels opkg-Paketmanager von OpenWrt wie folgt installiert werden.

```
$ opkg update
$ opkg install iptables-mod-tee
```

Listing 5.1: iptables Modul Installation

Die Konfiguration des Port Mirrorings erfolgte dann über zwei Aufrufe des iptables Programms mittels des zuvor installierten TEE Moduls. Wobei als *[IPADDRESS]* die IP-Adresse des Monitoring Servers angegeben wurde.

```
$ iptables -t mangle -A PREROUTING -j TEE --gateway [IPADDRESS]
$ iptables -t mangle -A POSTROUTING -j TEE --gateway [IPADDRESS]
```

Listing 5.2: Port Mirroring Konfiguration

Es werden somit zwei neue Regeln mittels `iptables` erstellt. `-t mangle` gibt die mangle Tabelle als Ziel an, welche für die Veränderung von Paketen verwendet wird. `-A PREROUTING` fügt die Regel der PREROUTING Kette (Chain) hinzu und sorgt damit dafür, dass alle Pakete vor dem Routing verarbeitet werden. `-A POSTROUTING` erfasst alle Pakete nachdem sie geroutet wurden. `-j TEE` sorgt dafür, dass das TEE Modul als Ziel gewählt wird und `--gateway [IPADRESS]` gibt an, wohin die Pakete weitergeleitet werden sollen.

Die Dokumentation des TEE Moduls besagt dazu:

```
TEE
The TEE target will clone a packet and redirect this clone to
   another machine on the local network segment. In other words, the
    nexthop must be the target, or you will have to configure the
   nexthop to forward it further if so desired.

--gateway ipaddr
    Send the cloned packet to the host reachable at the given IP
       address. Use of 0.0.0.0 (for IPv4 packets) or :: (IPv6) is
       invalid.

To forward all incoming traffic on eth0 to an Network Layer logging
   box:

-t mangle -A PREROUTING -i eth0 -j TEE --gateway 2001:db8::1
```

Nach der Konfiguration des Port Mirrorings wird nun der Datenverkehr des Routers als Kopie an den Monitoring Server geleitet und kann dort weiterverarbeitet werden.

5.1.2 Monitoring Server Konfiguration

Der Monitoring Server übernimmt in diesem Testaufbau gleich mehrere Rollen, im Einzelnen die der Datenerfassung, Datenhaltung sowie der Datenauswertung. Für den Teil der Datenerfassung wird wie bereits in Abschnitt 4.1 beschrieben die Software Packetbeat verwendet. Die Datenhaltung geschieht wie in Abschnitt 4.2 dargestellt mittels Elasticsearch und die Datenverarbeitung (4.3) wird anhand einer Python Anwendung auf Basis von Scikit-Learn (3.3.4) durchgeführt. Die Konfiguration der Komponenten auf dem Monitoring Server soll im Folgenden genauer betrachtet werden.

Packetbeat

Der gespiegelte Datenverkehr bzw. die einzelnen Pakete werden auf dem Server von dem Programm Packetbeat in der Version 5.2.1 in Empfang genommen. Die entsprechende Eingangs-Konfiguration sieht wie folgt aus:

```
1 # Select the network interface to sniff the data. On Linux, you can
    use the
2 # "any" keyword to sniff on all connected interfaces.
3 packetbeat.interfaces.device: any
```

Listing 5.3: Ausschnitt packetbeat.yml — Eingangs Konfiguration

Hier wird das Interface festgelegt, von welchem aus die Daten in Empfang genommen werden sollen. Anschließend erfolgt die Flow Konfiguration. Packetbeat kann den Datenverkehr schon vorab in sogenannte Flows zusammenfassen, wobei ein Flow der Übertragung mehrerer Datenpakete zwischen zwei Kommunikationspartnern des selben Kanals (Protokoll und Port) über einen definierten Zeitraum entspricht. Flows wurden in Abschnitt 4.1 bereits genauer erläutert. Im Flows-Abschnitt der Packetbeat Konfiguration kann eingestellt werden, wie lang der Zeitraum für die Zusammenfassung einzelner Pakete zu einem Flow maximal sein darf.

```
1 # Set ‘enabled: false‘ or comment out all options to disable flows
    reporting.
2 packetbeat.flows:
```

```
3  # Set network flow timeout. Flow is killed if no packet is
       received before being
4  # timed out.
5  timeout: 30s
6
7  # Configure reporting period. If set to -1, only killed flows will
       be reported
8  period: 10s
```

Listing 5.4: Ausschnitt packetbeat.yml — Flow Konfiguration

Darüber hinaus muss noch bestimmt werden, wohin die Ausgaben von Packetbeat hingeleitet werden sollen. Packetbeat bietet hier die Möglichkeit, aus verschiedenen Ausgabeformaten zu wählen. In diesem Fall soll die Ausgabe an eine Elasticsearch Instanz geschickt werden. Dabei gibt es die Möglichkeit einen oder mehrere Hosts sowie den verwendeten Port auszuwählen. Die Elasticsearch Instanz läuft in dem von dieser Arbeit verwendeten Aufbau ebenfalls auf dem Monitoring Server, daher wird `localhost` als Host und der Standard Port von Elasticsearch `9200` angegeben.

```
1 output.elasticsearch:
2  # Array of hosts to connect to.
3  hosts: ["localhost:9200"]
```

Listing 5.5: Ausschnitt packetbeat.yml — Ausgabe Konfiguration

Packetbeat wurde damit so konfiguriert, dass der Datenverkehr erfasst, in sogenannten Transaktion in Beziehung zu einander gebracht und anschließend direkt in Elasticsearch eingefügt wird.

Elasticsearch

Bei der Entwicklung des Prototypen wird wie im Konzept unter dem Abschnitt 4.2 vorgestellt Elasticsearch zur Speicherung der Daten eingesetzt. Elasticsearch ist ein Suchserver auf Basis der Apache Lucene Bibliothek, welcher auch als NoSQL-Datenbank eingesetzt wird.

Die Einrichtung des Elasticsearch Servers erfolgt in diesem Testaufbau über die Docker[2] Plattform. Hierzu wurde ein Docker Container des Elasticsearch Images in der Version 5

[2] https://www.docker.com/what-docker

verwendet, welches offiziell von Elasticsearch auf dem sogenannten Docker Hub[3] zur Verfügung gestellt wird. Der Elasticsearch Server wird dabei so Konfiguriert, dass er auf dem Standardport 9200 Anfragen über die REST API entgegen nimmt und Zugriffe von anderen Teilnehmern des lokalen Netzwerkes erlaubt. Grundsätzlich ist Elasticsearch ein verteiltes System und erlaubt die Aufteilung auf mehrere Instanzen, die sogenannten Nodes, welche wiederum in Shards aufgeteilt werden können. Innerhalb des Testaufbaus wird ein Master Node eingerichtet.

Das zuvor konfigurierte Packetbeat bietet eine direkte Anbindung an Elasticsearch und übergibt den korrelierten Datenverkehr in Form einzelner Dokumente im JSON Format über die REST API. Somit werden sämtliche Dokumente in Elasticsearch sofort indiziert und mit Metadaten angereichert. Jedes Dokument wird dabei einem passenden Typ zugeordnet. Mögliche Typen entsprechen dabei der Art des Vorgangs wie beispielsweise HTTP, MySQL, ICMP oder auch Flow. Diese werden dann zu einem Index mit der Bezeichnung *packetbeat* zusammengefasst. Wurde der Datenverkehr einmal von Elasticsearch indexiert, können beinahe in Echtzeit Abfragen des Datenbestandes durchgeführt werden. Elasticsearch ist damit für den Testaufbau fertig konfiguriert.

5.1.3 IP-Kamera Konfiguration

Die IP-Kamera spielt in diesem Testaufbau eine zentrale Rolle. Sie stellt beispielhaft eines der diversen IoT Endgeräte im lokalen Netzwerk dar und soll anhand der Security-Monitoring Software auf Verhaltensveränderungen überwacht werden. Bei der IP-Kamera handelt es sich um das Modell LogiLink WC0030A, welche über einen Bildsensor mit 0.3 MP verfügt. Die Anbindung an das Netzwerk erfolgt über einen kabelgebundenen Ethernet-Port oder drahtlos per WiFi, wobei die Kamera ausschließlich den IPv4 Standard beherrscht — eine Anbindung über IPv6 ist dabei nicht möglich. Außerdem verfügt die Kamera über ein Mikrofon und einen Lautsprecher sowie an der Vorderseite über einige Infrarot LEDs, welche für eine Nachtsichtfunktion verwendet werden können. Hinzu kommt die Möglichkeit, die Kamera per Fernsteuerung zu drehen und zu schwenken. Für die Fernsteuerung ist der Zugriff per Browser über ein Webinterface sowie eine Android Smartphone App möglich.

Für die Konfiguration der Kamera muss diese zunächst gestartet und über ein Netzwerkkabel mit dem Router verbunden werden. Nach dem ersten Startvorgang kann per Smartphone App über den Punkt *Add Camera* der QR-Code, welcher sich unter der Kamera befindet, gescannt

[3] https://hub.docker.com/_/elasticsearch/

werden, um das Smartphone mit der Kamera zu verbinden. Im Anschluss kann die Kamera über die App konfiguriert werden, so wird hier das WLAN des Routers in den Einstellungen ausgewählt und das Passwort eingegeben. Die Kamera startet im Anschluss neu und verbindet sich automatisch mit dem WiFi des Routers, eine Kabelverbindung ist dann nicht mehr nötig. Die Konfiguration der Kamera ist damit zunächst abgeschlossen.

5.1.4 Botnet und Command-and-Control-Server

In diesem Abschnitt soll die Konfiguration des Test-Botnetzes beschrieben werden. Die Bot-Software soll auf der IP-Kamera zur Ausführung gebracht werden, um Schaddaten im lokalen Netzwerk zu erzeugen.

Als Botnet bezeichnet man einen Verbund von mehreren Systemen für zumeist destruktive und verbrecherische Aktivitäten. Die einzelnen Systeme wurden dabei oftmals Opfer eines digitalen Einbruchs und der nachfolgenden Infektion mit einer speziellen Botnet-Software. Die infizierten Systeme, auch oft Bots oder Zombies genannt, verbinden sich mit einem Command-and-Control-Server (C&C-Server) und warten auf Steuerungsbefehle. Der C&C-Server ist dabei die zentrale Steuerungseinheit des Botnets, welcher die Bots verwaltet [SSPS13]. Die Steuerungsbefehle werden dabei oftmals von einem sogenannten Botmaster gegeben, welcher Zugriff auf den C&C-Server hat. Die Bots können dann beispielsweise dazu verwendet werden, Distributed Denial of Service-Attacken (DDoS-Attacken) auf andere Systeme zu starten, indem eine große Anzahl der Bots gleichzeitig Zugriffe auf das vom C&C-Server genannte Ziel des Angriffs durchführen. Abhängig von der Größe des Botnets und der auf dem Zielserver getroffenen Schutzvorkehrungen kann so ein Angriff den Ausfall des Systems zur Folge haben, da die Masse der Anfragen nicht mehr bewältigt werden kann [LJZ09; FSR09]. Andere Botnets werden dazu verwendet große Mengen an Spam-Mails zu versenden oder um die Rechenleistung der Bots für das Schürfen von Bitcoins zu nutzen. Die Abbildung 5.2 stellt die einzelnen Elemente eines Botnets grafisch dar.

Die in diesem Testaufbau verwendete Botnet-Software ist bekannt unter dem Namen Mirai-Botnet und ist mit besonderem Fokus auf das IoT entwickelt worden. Der Quellcode dieser Software ist durch einen Forenbeitrag des angeblichen Entwicklers an die Öffentlichkeit gelangt und steht seitdem auch der Forschung über diverse Code-Hosting Dienste zur Verfügung. Aus dem Quellcode ist ersichtlich, dass die Software nach Geräten im Internet sucht und versucht, sich an diesen Geräten mittels einer Liste von Standardpasswörtern per Telnet anzumelden.

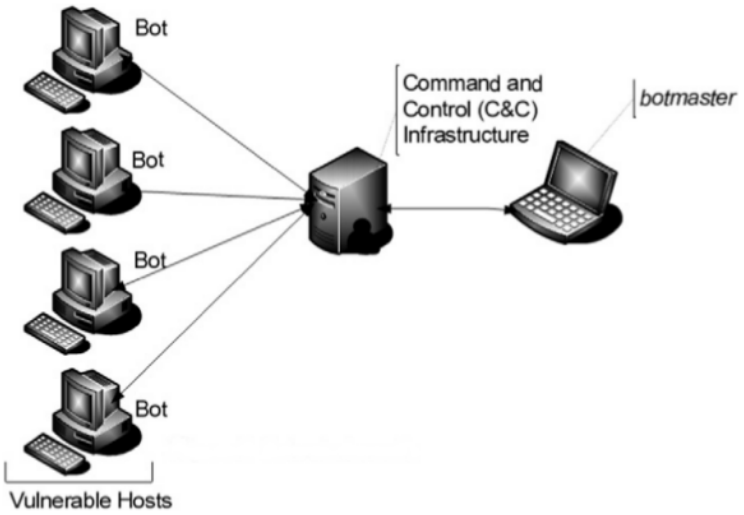

Abbildung 5.2: Elemente eines Botnets [SSPS13]

Gelingt dies, wird die Software über einen Loader-Server auf dieses Gerät gespielt und dort gestartet. Alle Geräte auf denen diese Software läuft, verbinden sich mit dem C&C-Server und können von diesem Befehle für gezielte DDoS-Angriffe auf andere Systeme erhalten. Die hier im Testaufbau verwendete IP-Kamera weist genau diese Schwachstelle eines schwachen Standardpassworts auf und ist damit anfällig für diese Art der Infektion über das Mirai-Botnet.

Für die Simulation einer erfolgreichen Infektion der IP-Kamera durch das Mirai-Botnet, wurde anhand des veröffentlichten Quellcodes ein eigener C&C-Server aufgesetzt. Aus Gründen der Isolation vom Hostsystem und der Flexibilität, wurde für den Betrieb des C&C-Servers ein Docker-Container verwendet, welcher außerdem innerhalb eines eigenen Subnetzes liegt. Dies soll sicherstellen, dass die Ergenisse nicht dadurch verfälscht werden, dass sich sowohl die Kamera welche als Bot fungiert, als auch der C&C-Server im gleichen Netz befinden. Dieser Umstand würde in reellen Botnet Struktur nur selten tatsächlich auftreten. In diesem Container wurde die Konfiguration durchgeführt sowie der Quellcode kompiliert. Der Betrieb des Servers setzt außerdem eine MySQL-Datenbank für die Verwaltung der Bots voraus, welche ebenfalls in dem Docker-Container aufgesetzt wurde. Anschließend wurden die benötigten Datenbanken erzeugt, sowie ein neuer Benutzer samt Passwort in der Benutzer-Tabelle angelegt. Mittels dieser Benutzerdaten ist es dann möglich, sich an dem C&C-Server zu authentifizieren. Im Anschluss kann der C&C-Server gestartet werden und ist damit betriebsbereit. Er wartet von nun an auf die Verbindungen der Bots und kann die Befehle des Botmasters entgegennehmen.

Die Bot-Software, welche auf der Kamera zur Ausführung gebraucht werden soll, wurde ebenfalls in diesem Schritt für die entsprechende Prozessorarchitektur der Kamera mittels eines Cross-Compilers erzeugt. In diesem Falle handelte es sich dabei um eine MIPS-Architektur (**M**icroprocessor without **i**nterlocked **p**ipeline **s**tages). Dabei wurden zuvor der Quellcode der Bot-Software so konfiguriert, dass die IP-Adresse des C&C-Servers fest hinterlegt ist. Im Normalfall würde der Bot die Adresse des C&C-Servers anhand einer Domain per DNS (Domain Name System) ermitteln. Die ausführbare Bot-Software wurde somit konfiguriert und erzeugt und ist somit bereit für den Einsatz auf der Kamera.

5.2 Implementierung des Prototypen

Angelehnt an das in Abschnitt 4 dargestellte Konzept der Softwarearchitektur, wurden die Aspekte der Datenerfassung, Datenhaltung und Datenverarbeitung beispielhaft umgesetzt. Die entsprechende Implementierung wird nachfolgend dargestellt.

5.2.1 Datenerfassung und Datenhaltung

Für die Implementierung des Prototypen sind möglichst valide Testdaten von essentieller Bedeutung. Das bedeutet, dass die Testdaten von einem echten IoT Gerät stammen und somit auf einen vergleichbaren produktiven Anwendungsfall übertragen werden können. Mit Hilfe des in Abschnitt 5.1.1 konfigurierten Port-Mirrorings ist Packetbeat in der Lage, die von der Kamera produzierten Daten zu erfassen. Die Datenerfassung teilt sich hierbei in zwei Schritte auf. Zum einen die Erfassung der Kommunikationsdaten, welche die Kamera produziert solange sie nicht unter dem Einfluss von Schadsoftware steht. Also die Art von Daten, die aufkommen wenn die Kamera beispielsweise auf eine Verbindung eines Nutzers wartet oder wenn ein Nutzer die Kamera gerade bedient. Diese werden hier als Normdaten bezeichnet. Zum anderen werden Daten erfasst, die das Gerät produziert, wenn es unter dem Einfluss von Schadsoftware, der sogenannten Malwar, steht. Diese werden hier als bösartige Daten bezeichnet. Die Erfassung der Normdaten sowie der bösartigen Daten findet zeitlich getrennt voneinander statt, damit diese zu einem späteren Zeitpunkt unabhängig voneinander betrachtet und weiterverarbeitet werden können.

Um nun die Normdaten zu gewinnen, wurde die IP-Kamera gestartet und mit dem lokalen Netzwerk verbunden. Anschließend wurde das Programm Packetbeat gestartet, um den Daten-

verkehr aufzuzeichnen. Von nun an wurde für etwa eine Stunde der Datenverkehr des Netzes aufgezeichnet, wobei zwischenzeitlich häufiger per Smartphone App eine Verbindung mit der Kamera aufgebaut wurde. Dabei wurden verschiedene Funktionen der Kamera genutzt, wie beispielsweise das Schwenken und Neigen des Kamerakopfes sowie die Nutzung des Kamera-internen Mikrofons und Lautsprechers, um bestmöglich eine normale Nutzung zu simulieren. Nach etwa einer Stunde Normalbetrieb der Kamera, wurde die Aufzeichnung der Normdaten abgeschlossen.

Zur Erzeugung der bösartigen Daten wurde die in Abschnitt 5.1.4 vorgestellte Mirai-Bot-Software verwendet, welche auf der IP-Kamera gestartet wurde. Im Falle der in dieser Arbeit verwendeten IP-Kamera, lag unter anderem die in Punkt 3.2.2 angesprochene Schwachstelle eines schwachen Standardpassworts vor. Der Root-Zugriff war mit dem Passwort „123456“ per Telnet möglich. Dieses Passwort ließ sich zudem auch nachträglich nicht über die Benutzeroberfläche der Steuerungssoftware ändern. Um den Root-Zugriff auf diese Kamera per Standardpasswort zu verhindern, sind also zumindest rudimentäre Linux-Kenntnisse sowie die Kenntnis dieses vom Hersteller undokumentierten Zugangs nötig. Zudem wird das Passwort nach einem Neustart der Kamera wieder auf den Standardwert zurückgesetzt. Eine dauerhafte Änderung ist so also nicht möglich. Dieser Umstand wurde im Rahmen der Implementierung des Prototypen genutzt, um die Mirai-Bot-Software auf die Kamera zu übertragen. Die Bot-Software wurde zuvor wie in Abschnitt 5.1.4 beschrieben für die Prozessorarchitektur der Kamera erzeugt. Nach der Erzeugung wurde das Programm Packetbeat ein weiteres Mal auf dem Monitoring-Server gestartet um den Datenverkehr aufzuzeichen. Anschließend wurde die Bot-Software mittels Trivial File Transfer Protocol (TFTP) auf die IP-Kamera übertragen und zur Ausführung gebracht. Ein Angriff wurde in dieser Zeit nicht vom C&C-Server initiiert. Es wurden somit lediglich Daten aufgezeichnet, welche die Bot-Software produziert, während sie auf Befehle wartet. Nach einigen Minuten wurde Packetbeat gestoppt und die Aufzeichnung der bösartigen Daten damit abgeschlossen.

Packetbeat hat somit die von der Kamera produzierten Normdaten sowie die bösartigen Daten aufgezeichnet und an Elasticsearch zur Datenhaltung weitergeleitet.

5.2.2 Datenverarbeitung

Im folgenden Abschnitt soll anhand von Codebeispielen gezeigt werden, wie die Datenverarbeitung des in Abschnitt 4.3 vorgestellten Konzeptes mittels des beschriebenen Knowledge

Discovery Process (??) anhand eines Prototypen umgesetzt werden kann. Dabei erfolgte die Datenverarbeitung im Rahmen dieses Prototypen mittels eines in der Programmiersprache Python geschriebenen Programms. Verwendet wurde hierbei Python in der Version 3.5.3, mit den wie in Abschnitt 3.3.4 beschriebenen Paketen *numpy 1.12.0*, *scipy 0.18.1* und *scikit-learn 0.18.1*. Für die Anbindung dieses Programms an die REST API von Elasticsearch wurde das Paket *elasticsearch 5.2.0* eingesetzt. Zunächst wurde eine **Selektion** der Daten vorgenommen, wobei eine Abfrage an die Datenhaltung gestellt werden musste. In Listing 5.6 ist zu sehen, wie die Normdaten in Form der Query DSL von Elasticsearch angefragt wurden. Dabei wurden zunächst die Verbindungsdaten zu Elasticsearch angegeben und dann eine Abfrage erstellt. Die Abfrage setzte sich aus zwei Teilen zusammen, wobei der erste bestimmt, dass die Daten vom Typ *flow* sein müssen und als Sender oder Empfänger die IP-Adresse der Kamera (192.168.1.206) enthalten sein muss. Der zweite Teil bestimmt, aus welchem Zeitraum die Daten stammen, hier wird der Zeitraum angegeben, in dem die Aufzeichnung der Normdaten stattgefunden hat.

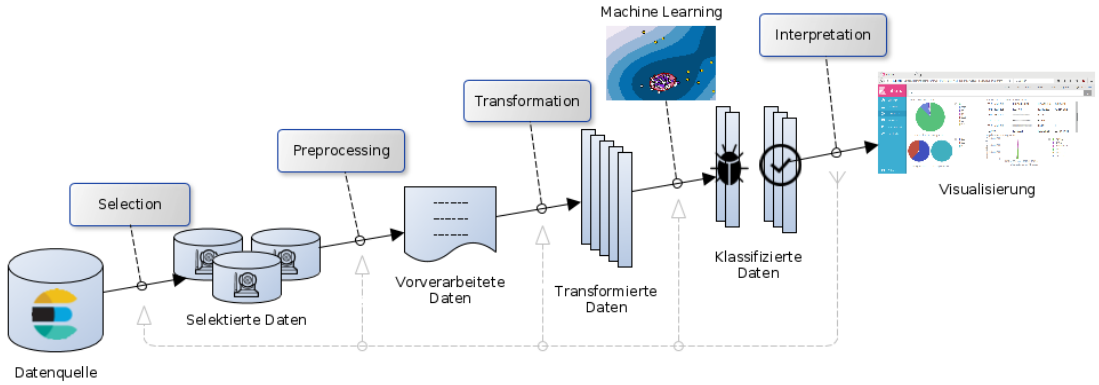

Abbildung 5.3: Knowledge Discovery Process [nach FPSS96a, S. 41]

```
from elasticsearch import Elasticsearch

es = Elasticsearch([{'host': 'localhost', 'port': 9200}])

clean_result = es.search(
        index='packetbeat-*',
        body={
          'query': {
            'bool': {
              'must': [
                {
```

```
                    'query_string': {
                      'query': 'type:flow AND "192.168.1.206"'
                    }
                  },
                  {
                    'range': {
                      'start_time': {
                        'gte': '2017-03-07T00:00:00.000Z',
                        'lte': '2017-03-08T17:30:00.000Z'
                        }
                      }
                    }
                  ]
                }
              }
            },
          )
```

Listing 5.6: Selektion der Daten

Das Ergebnis der Anfrage kommt in Form eines JSON Dokuments von Elasticsearch zurück. Bevor die Daten jedoch weiterverwendet werden können, müssen sie zunächst **vorverarbeitet** (preprocessing) und in ein Array von Featuresets **transformiert** werden. Wobei die Vorverarbeitung und die Transformation in diesem Prototypen direkt in einem Schritt durchgeführt werden. In Listing 5.7 ist zu sehen, wie die Suchergebnisse in Form der JSON Dokumente *clean_result* umgewandelt und der Variable *X_norm* zugewiesen werden.

```
X_norm = transform_to_featuresets_array(clean_result)

X_train = X_norm[:799]
X_test = X_norm[-533:]
```

Listing 5.7: Transformation der Daten

In Listing 5.8 wird im Detail gezeigt wie alle Flow-Datensätze, in ein Array transformiert werden.

```
def transform_to_featuresets_array(elastic_results):
    all_featuresets = []
    for obsrv in elastic_results['hits']['hits']:
        featureset = transform_flow_to_featureset(obsrv)
```

```
5         if featureset:
6             all_featuresets.append(featureset)
7     return np.array(all_featuresets)
```

Listing 5.8: Transformation aller Flow-Datensätze zu einem Array

Listing 5.9 zeigt die Vorgehensweise der Umwandlung eines einzelnen Flow-Datensatzes in eine Liste von Features. Die einzelnen Features werden hier außerdem normalisiert. In Zeile 6 ist beispielsweise zu sehen, wie die IP-Adresse des Senders zunächst mittels der *split_ip*-Funktion in die einzelnen durch Punkte getrennten Blöcke zerlegt wird. Anschließend werden diese Blöcke einzeln normalisiert und jeder der vier Blöcke für die IP-Adresse des Senders, wird dann als eigenes Feature in Zeile 19–22 dem Featureset hinzugefügt. Genau so wird auch mit der IP-Adresse des Empfängers verfahren.

```
1  def transform_flow_to_featureset(obsrv):
2      obsrv_src = obsrv['_source']
3      if not is_flowdata_complete(obsrv_src):
4          return
5
6      norm_split_src_ip = normalize_ip(split_ip(obsrv_src['source']['
           ip']))
7      norm_split_dest_ip = normalize_ip(split_ip(obsrv_src['dest']['ip
           ']))
8
9      src_port = obsrv_src['source']['port']
10     dest_port = obsrv_src['dest']['port']
11
12     src_packets_total = obsrv_src['source']['stats']['
           net_packets_total']
13     src_bytes_total = obsrv_src['source']['stats']['net_bytes_total'
           ]
14
15     dest_packets_total = obsrv_src['dest']['stats']['
           net_packets_total']
16     dest_bytes_total = obsrv_src['dest']['stats']['net_bytes_total']
17
18     return [
19             norm_split_src_ip[0],
20             norm_split_src_ip[1],
21             norm_split_src_ip[2],
```

```
            norm_split_src_ip[3],
            src_port,
            src_packets_total / max_src_packets,
            src_bytes_total / max_src_bytes,
            norm_split_dest_ip[0],
            norm_split_dest_ip[1],
            norm_split_dest_ip[2],
            norm_split_dest_ip[3],
            dest_port,
            dest_packets_total / max_dest_packets,
            dest_bytes_total / max_dest_bytes
            ]
```

Listing 5.9: Transformation eines einzelnen Flow-Datensatzes in ein Featureset

Listing 5.10 zeigt dabei noch einmal genau wie eine IP-Adresse zunächst in ihre Blöcke zerlegt und anschließend jeder einzelne Block normalisert wird. Für den Prozess der Normalisierung wird die Zahl des Adressblockes durch 255 geteilt, da jeder Block einer IPv4-Adresse in einem Bereich von 0–255 liegen muss. Daraus entsteht ein normalisertes numerisches Array einer IP-Adresse.

```
def split_ip(ip_adr):
    return [int(x) for x in ip_adr.split('.')]


def normalize_ip(ip_splitted):
    return [x / 255 for x in ip_splitted]
```

Listing 5.10: Normalisierung der IP-Adressen

Wie in Listing 5.7 bereits zu sehen war, wurden die Featuresets nach der Transformation in die zwei Variablen *X_train* und *X_test* aufgeteilt. Dabei wurden die ersten 799 Featuresets der Flowdaten in *X_train* und die letzten 533 in *X_test* gespeichert. Da die Gesamtanzahl aller Normdaten in *X_norm* 1332 Datensätze enthält, handelt es sich somit dabei um zwei disjunkte Mengen. Die Menge an Featuresets aus *X_train* wurde dann dazu verwendet, den Machine Learning Algorithmus zu trainieren. Dies geschieht über die in Listing 5.11 dargestellte *fit*-Funktion der Klasse One-Class SVM aus dem Paket Scikit-Learn. Bei der Instanziierung der One-Class SVM sind die Werte nu und $gamma$ von besonderer Relevanz. Der nu-Parameter erlaubt Werte von 0–1 und gibt den zu erwartenden Anteil von Fehlern in den Trainingsdaten

an. Ein Wert von beispielsweise 0.2 würde einer Rate von 20% Schaddaten in den Trainingsdaten entsprechen. Für die hier genutzte One-Class SVM wurde ein niedriger nu-Wert gewählt, da davon auszugehen ist, dass die Trainigsdaten hauptsächlich aus nicht infiziertem Datenverkehr bestehen. Der $gamma$-Wert liegt zwischen 0–1, beeinflusst die Kernel Funktion und bestimmt, wieviel Einfluss ein einzelner Datensatz auf die Hyperebene hat. Ein hoher Wert grenzt die Trainingsdaten sehr scharf ab und läuft damit Gefahr, alle Datensätze, die eine geringe Abweichung von den Trainingsdaten aufweisen, als Schaddaten zu klassifizieren. Für die One-Class SVM wurde dementsprechend ein niedriger Wert ausgewählt, um eine gewisse Toleranz in den nicht schadhaften Daten zu erlauben.

```
1 from sklearn import svm
2 clf = svm.OneClassSVM(nu=0.01, kernel="rbf", gamma=0.0000001)
3 clf.fit(X_train)
```

Listing 5.11: Anlernen des Machine Learning Algothimus

Der One-Class SVM ist an dieser Stelle mit den Trainingsdaten trainiert worden und konnte anschließend verwendet werden, um Voraussagen beziehungsweise Einordnungen anderer Daten vorzunehmen. Dazu wurde wie in Listing 5.12 gezeigt, die *predict*-Funktion verwendet. Diese nimmt genau wie die *fit*-Funktion ein Array von Featuresets entgegen und gibt ein Array mit Ergebnissen zurück.

```
1 y_pred_train = clf.predict(X_train)
2 y_pred_test = clf.predict(X_test)
```

Listing 5.12: Voraussagen über die Train- und Testdaten

Da es sich um eine One-Class SVM handelt, kann der Rückgabe-Wert für jedes Featureset entweder 1 oder -1 sein. 1 bedeutet in diesem Fall, dass das Featureset des Flows als „Normal“ eingestuft wurde. -1 bedeutet hingegen, dass das Featureset als bösartig eingestuft wurde. Nach diesem beschriebenen Vorgehen lassen sich auch die erzeugten bösartigen Daten (X_{mixed}) über Elasticsearch selektieren, transformieren und über die *predict*-Funktion auswerten. Da sich dieses Vorgehen bis auf den veränderten Zeitraum in der Selektion der Daten nicht von dem gezeigten unterscheidet, wird auf die Quellcode-Ausschnitte dafür an dieser Stelle nicht näher eingegangen.

Zusammenfassend wurde in diesem Abschnitt anhand von Codebeispielen gezeigt, wie nach dem Prinzip des Knowledge Discovery Process die Flowdaten der Kamera aus der Datenhaltung selektiert, transformiert und ausgewertet werden können. Als Ergebnis gibt die Einheit der

Datenverarbeitung eine Vorhersage für jeden einzelnen Flow ab, wobei dieser entweder in der Norm liegen kann oder bei Abweichung davon als bösartiger Datenverkehr eingestuft wird.

6 Ergebnisse

Nachdem im vorangegangenen Abschnitt anhand eines Prototypen gezeigt wurde, wie mittels eines intelligenten Systems Security-Monitoring für IoT Endgeräte in einem lokalen Netzwerk betrieben werden kann, werden im Folgenden die Ergebnisse der Datenverarbeitung dieses Prototypen dargestellt und grafisch aufbereitet. Es handelt sich dabei um die Ergebnisse der Anwendung eines Machine Learning Verfahrens, der sogenannten One-Class SVM, auf den Datenverkehr einer IP-Kamera, welche beispielhaft ein IoT Endgerät innerhalb eines lokalen Netzwerkes repräsentiert. Die daraus gewonnen Ergebnisse sollen darüber hinaus als Teil einer Security-Monitoring Software für den Nutzer Aufschluss über den Zustand der IoT Geräte geben, also ob diese Geräte ihren *normalen* Aufgaben nachgehen oder ob sie möglicherweise mit einer Schadsoftware infiziert wurden.

Als Teil dieses Prozesses wurden zunächst wie in Abschnitt 5.2.1 beschrieben, die Normdaten der Kamera aufgezeichnet. Also der Datenverkehr, welchen die Kamera ohne Einfluss einer Schadsoftware generiert. Diese gesamte Menge der Normdaten wird hier als X_{norm} bezeichnet. Ein Teilmenge der Daten von X_{norm} wurde zum Trainieren der One-Class SVM verwendet. Diese Daten bezeichnen wir hier als X_{train}. Eine weitere Teilmenge von X_{norm} wurde als Test der Vorhersage-Funktion der One-Class SVM verwendet, diese Daten nennen wir hier X_{test}. Dabei ist festzuhalten, dass sich die Teilmengen X_{train} und X_{test} nicht überschneiden. X_{train} ist also disjunkt zu X_{test}.

Die Erfassung der Normdaten nach dem in Abschnitt 5.2.1 beschriebenen Verfahren hat nach der Selektion eine Anzahl von insgesamt 1333 Flowdaten ergeben. Daraus ergibt sich $|X_{norm}| = 1333$. Dabei ist festzuhalten, dass es sich dabei nicht um einzelne Datenpakete handelt, sondern um sogenannte Packetbeat Flows, welche wie in Abschnitt 5.1.2 konfiguriert wurden, womit diese mehrere Datenpakete zu einem Flow zusammenfassen. X_{norm} wurde nun in zwei Teile X_{train} und X_{test} aufgeteilt, wobei ein Verhältnis von 60/40 gewählt wurde. Daraus ergibt sich $|X_{train}| = 799$ und $|X_{test}| = 533$. Die erfassten Daten des schadhaften Datenverkehrs unter Einfluss des Botnets nach dem in Abschnitt 5.2.1 beschriebenen Verfahren

werden nachfolgend als X_{mixed} bezeichnet. Diese enthalten einen Mix aus schadhaften Daten und Normdaten. Hier ergab sich nach der Selektion eine Anzahl von 10831 Flowdaten. Es gilt also $|X_{mixed}| = 10831$.

X_{train} wurde nun zunächst verwendet um die One-Class SVM zu trainieren. Im Anschluss wurde die Präzision der Voraussage anhand von Tests überprüft. Dabei wurde die One-Class SVM im ersten Schritt mit den gleichen Daten getestet, mit welchen sie auch trainiert wurde, also mit X_{train}. Die One-Class SVM hat dabei 792 von insgesamt 799 Flows als nicht schadhaft und wiederum 7 davon als schadhaft beziehungsweise außerhalb der Norm eingestuft. Im nächsten Schritt wurde die SVM mit den Daten aus X_{test} sowie X_{mixed} getestet. Ordnet die SVM einen Flow als schadhaft ein, dann wird er als *positive* bezeichnet. Wird der Flow als nicht schadhaft eingeordnet, dann wird er als *negative* bezeichnet. Außerdem werden korrekt als schadhaft eingeordnete Daten als *true-positives* (TP) und korrekt als nicht schadhaft eingeordnete Daten als *true-negatives* (TN) bezeichnet. Wohingegen fälschlich als schadhaft eingeordnete Daten als *false-positives* (FP) und fälschlich als nicht schadhaft eingeordente Daten als *false-negatives* (FN) bezeichnet werden. Die Einordnung der Daten aus X_{mixed} über die Voraussage-Funktion der One-Class SVM ist in Abbildung 6.1 anschaulich dargestellt.

		tatsächlicher Zustand	
		Schaddaten	keine Schaddaten
One-Class SVM Voraussage	als Schaddaten eingeordnet	TP 9979 (92,1%)	FP 90 (0,8%)
	nicht als Schaddaten eingeordnet	FN 19 (0,2%)	TN 743 (6,9%)

Abbildung 6.1: Einordnung der Voraussage Daten von X_{mixed}

Aus diesen Daten lassen sich wiederum die Werte *Precision*, *Recall* und *F1 Score* wie in Abblidung 6.2 dargestellt berechnen.

Als Precision (deutsch: Genauigkeit) bezeichnet man den Anteil der tatsächlichen Schaddaten an allen als Schaddaten eingeordneten Daten. Der Wert gibt demzufolge die Wahrscheinlichkeit an, mit dem eine Einordnung in die Gruppe der Schaddaten zutreffend ist. Unter Recall (deutsch: Trefferquote) versteht man die Wahrscheinlichkeit, mit der Schaddaten tatsächlich als solche erkannt werden. Es handelt sich also um den Anteil der korrekt eingeordneten Schaddaten. Mittels des F-Maßes ($F_1 Score$) wiederum kann ein gewichtetes harmonisches Mittel von Precision und Recall bestimmt werden. Die Abbildung 6.3 gibt die Ergebnisse für

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Abbildung 6.2: Berechnung von Precision, Recall und F1-Score

den konkreten Anwendungsfall wieder. Betrachtet werden hier die Daten von X_{mixed}, da sie sowohl Schaddaten als auch normale Daten enthalten und demzufolge für die Berechnung von Precision, Recall und F1 genutzt werden können.

Datensatz	Anzahl	true-positives	true-negatives	false-positives	false-negatives	Precision	Recall	F1
X_{train}	799	0	792	7	0	0	0	0
X_{test}	533	0	448	85	0	0	0	0
X_{mixed}	10831	9979	743	90	19	0,991	0,997	0,994

Abbildung 6.3: Voraussage-Ergebnisse der SVM

Es zeigt sich, dass die Precision im Anwendungsfall bei 99,1% liegt. Das bedeutet, dass es sich bei den vom Algorithmus als Schaddaten eingeordneten Daten, zu 99,1% tatsächlich um Schaddaten handelt. Der Recall liegt im vorliegenden Fall bei 99,7%. Das bedeutet, dass Schaddaten zu 99,7% als solche erkannt werden. Es gab 90 FPs, was einem Anteil von 0,8% entspricht.

Zusammenfassend kann man anhand der Daten in Abbildung 6.3 ablesen, dass sich die von der Botsoftware produzierten Daten in X_{mixed} mittels des Machine Learning Verfahrens gut automatisch erkennen lassen. Eine Diskussion der hier präsentierten Ergebnisse erfolgt in Abschnitt 7.

7 Diskussion

In der vorliegenden Arbeit wurde ein Konzept vorgestellt, welches es ermöglicht, Security-Monitoring von IoT Endgeräten in lokalen Netzwerken unter Einsatz eines intelligenten Systems durchzuführen. Wesentliche Teile des Konzepts wurden dabei als Prototyp mittels Machine Learning umgesetzt. Die dabei gewonnen Erkenntnisse sollen nachfolgend diskutiert werden.

Wie in Abschnitt 6 gezeigt wurde, konnte durch den Einsatz des Machine Learnings im Rahmen des Testaufbaus mit den mixed-Daten eine Genauigkeit (Precision) von 99,1% erreicht werden. Das bedeutet, dass es sich bei 99,1% der vom Algorithmus als Schaddaten eingeordneten Daten auch tatsächlich um Schaddaten handelte. Die Trefferquote (Recall) lag wiederum bei 99,7%. Das bedeutet, dass Schaddaten zu 99,7% auch als solche erkannt wurden. Setzt man Genauigkeit und Trefferquote mittels F1 ins Verhältnis, so ergibt sich ein Wert von 99,4%. Diese Ergebnisse sind als sehr gut einzustufen. Insbesondere die Trefferquote beziehungsweise Erkennungsrate von 99,7%, spricht dafür, dass die One-Class SVM erfolgreich zwischen normalen Daten und Schaddaten unterscheiden konnte.

Ein möglicher Grund für diese erfolgreiche Erkennung des schadhaften Datenverkehrs der verwendeten IP-Kamera durch den Algorithmus liegt darin begründet, dass sich die trainierten Daten des normalen Verhaltens deutlich von den Schaddaten nach der Infizierung unterscheiden. Da IoT Endgeräte häufig sehr spezielle Aufgaben übernehmen, ist davon auszugehen, dass sich die Normaldaten auch bei anderen Endgeräten deutlich von den nach einer Malware Infektion produzierten Schaddaten abgrenzen lassen. Diese Annahme sollte in zukünftigen Forschungsarbeiten zur IoT-Sicherheit überprüft werden. Dabei ist zu beachten, dass die hier verwendete Malware in Form des sogenannten Mirai-Botnets bereits ohne Steuerungsbefehle des C&C-Servers sehr auffällig agierte. Wie sich nach Analyse des Mirai-Quellcodes herausstellte, startete die Software direkt nach Ausführung auf der Kamera einen Suchvorgang nach weiteren potenziellen Infizierungsopfern im Internet mit zufälligen Adressen. Dabei wurden massive Anmeldeversuche über Telnet auf Port 23 durchgeführt, welche zu einer

besonderen Auffälligkeit im Datenverkehr führten und die Schadsoftware somit bereits zu diesem Zeitpunkt durch das Machine Learning erkennen ließ. Es bleibt also zu überprüfen, ob andere Schadprogramme ein ähnlich auffälliges Verhalten aufweisen und sich ebenfalls so gut von dem normalen Verhalten der IoT Endgeräte unterscheiden lassen.

Um eine genauere Wertung der genannten Ergebnisse (Precision, Recall, F1) vornehmen zu können, ist es sinnvoll, die sogenannte false positve-Rate noch einmal separat in die Betrachtung mit einzubeziehen. Im Anwendungsfall lag dieser Wert bei 0,8%. Das heißt, dass 0,8% der normalen Daten vom Algorithmus fälschlicherweise als Schaddaten eingeordnet wurden. Dieser Wert ist recht gering, was als positiv zu bewerten ist. Eine geringe FP-Rate ist besonders für ein Security-Monitoring System von zentraler Bedeutung, da das System zwar Schadsoftware möglichst gut erkennen soll aber gleichzeitig nicht ständig einen Fehlalarm produzieren darf. Winter (2011) geht dabei so weit, zu sagen: „*Für die Entwickler von Systemen zur Anomalieerkennung gilt, dass die FAR (Falschalarmrate) eines Systems bei praktisch 0% liegen muss. Aus diesem Grund sollten Systeme zur Anomalieerkennung immer in Hinsicht auf ihre FAR optimiert werden, obwohl dies die Erkennungsrate negativ beeinflussen kann.*“ [WLZH11, S. 237]. Folgt man dieser Annahme von Winter, so wäre es wünschenswert, die in der vorliegenden Arbeit erreichte FP-Rate weiter zu reduzieren. Neben der FP-Rate lassen auch die übrigen berichteten Kennwerte (Precision, Recall) einen gewissen Spielraum für Optimierungen. Um diese Optimierungen zu erreichen, gibt es verschiedene Parameter des One-Class SVM, die angepasst werden können. Ein Beispiel hierfür ist der Gamma-Wert, der bestimmt, wieviel Einfluss ein einzelner Flow in der Trainingsphase auf spätere Voraussagen hat. Auch die Verwendung verschiedener oder eigener SVM Kernels wäre in Erwägung zu ziehen, um die Ergebnisse weiter zu verbessern. Die Suche nach solchen optimierten Parametern wird auch als *hyperparameter optimization* oder *model selection* bezeichnet und kann durch den Einsatz von Algorithmen automatisiert werden.

Eine weitere Möglichkeit, die erzielten Ergebnisse zu verbessern, besteht in der Veränderung der Lernphase. So ist es wichtig, dass die Geräte innerhalb der Lernphase möglichst alle Aktivitäten zeigen, die im Normalbetrieb auftreten, damit ein umfassendes Abbild des normalen Datenverkehrs gewonnen werden kann. Um dieses Ziel zu erreichen, ist es essenziell, dass das Gerät während der Lernphase nicht bereits infiziert ist. Darüber hinaus kann es sinnvoll sein, die Zeit für den Lernvorgang zu erhöhen. In der vorliegenden Arbeit lag die Lernzeit beispielsweise bei rund einer Stunde, während der die Kamera mit ihren verschiedenen Funktionen genutzt wurde. Für zukünftige Arbeiten mit einem ähnlichen Ansatz könnte diese Lernzeit erweitert werden, um zu überprüfen, ob sich dadurch Verbesserungen der Ergebnisse erzielen lassen.

Neben den genannten Aspekten, die die Ergebnisse (Precision, Recall, F1) der vorliegenden Arbeit betreffen, gibt es weitere Ansatzpunkte, mögliche Veränderungen beziehungsweise Erweiterungen der genutzten Methodik vorzunehmen. Ein Aspekt betrifft die in der vorliegenden Arbeit genutzte Methode des Port Mirrorings zur Erfassung des Datenverkehrs. Im Rahmen des Testaufbaus stellte sich diese Methode als durchaus geeignet dar, da die Daten ohne zusätzliche Hardware gespiegelt und weitergeleitet werden konnten. Für Netzwerke mit einem sehr großen Datenaufkommen, kann diese Methode jedoch an ihre Grenzen stoßen. Der Grund dafür ist rein technischer Natur und liegt darin begründet, dass der gesamte Datenverkehr des Netzwerks über eine einzelne Leitung zur Datenerfassung übertragen werden muss. Übersteigt das Datenaufkommen des Netzwerks die Kapazität der Leitung kommt es zum Datenverlust. Die verlorenen Daten können somit nicht ausgewertet werden und fallen aus dem Security-Monitoring heraus. In solchen Fällen wäre es also durchaus sinnvoll, andere Methoden zur Erfassung des Datenverkehrs in Betracht zu ziehen.

Zusätzlich zu diesen Erkenntnissen, die sich aus den in der vorliegenden Arbeit erzielten Ergebnissen ableiten lassen, liefert auch die gewählte Herangehensweise im Rahmen der Implementierung interessante Erkenntnisse zum Thema der IoT-Sicherheit. So wurde bei der verwendeten IP-Kamera das schwache Standardpasswort ausgenutzt, um die Bot-Software zu installieren. Das Problem der schwachen Standardpasswörter scheint im IoT-Bereich häufig aufzutreten. Ein Hinweis hierfür ist das auch in dieser Arbeit genutzte Mirai-Botnet, welches in jüngster Zeit DDos-Angriffe auf verschiedene Ziele durchgeführt und dabei Werte von bis zu 1,2 Terabit pro Sekunde erreicht hat. Dieses Botnet bestand zu der Zeit der Angriffe (um den Oktober 2016) aus etwa einer Million infizierten Geräten [Ble17]. In Bezug auf die Sicherheit des IoT kann man festhalten, dass solide Kommunikationsstandards und Verschlüsselung zwar helfen können, die Sicherheit zu verbessern. Probleme wie schwache Standardpasswörter, ungeschütze Zugänge und fehlende Updatefunktionen umgehen diese Vorkehrungen allerdings [SBK16]. Ein Security-Monitoring für das IoT, wie es in dieser Arbeit beispielhaft dargestellt wurde, kann dabei dazu beitragen, erfolgreiche Angriffe zu erkennen.

Zusammenfassend zeigt sich, dass die vorliegende Arbeit eine erfolgreiche Möglichkeit aufzeigt, wie Security-Monitoring von IoT Endgeräten in einem lokalen Netzwerk mittels Machine Learning betrieben werden kann. Trotz der bereits sehr guten Ergebnisse hinsichtlich der Genauigkeit (Precision), der Trefferquote (Recall) sowie der false-positive-Rate, konnten Verbesserungsvorschläge hinsichtlich des verwendeten Algorithmus sowie bestimmter Methodik-Aspekte aufgezeigt werden. Diese Empfehlungen sollten in zukünftige Forschungsarbeiten zur IoT-Sicherheit einfließen.

8 Ausblick

Im vorangegangenen Abschnitt wurden bereits Empfehlungen für mögliche Verbesserungen der in dieser Arbeit gewählten Vorgehensweise aufgezeigt. So wurde deutlich gemacht, dass die gewonnen Ergebnisse beispielsweise durch die Optimierung von Parametern der One-Class SVM sowie die Verlängerung der Lernphase weiter verbessert werden könnten. Darüber hinaus wurde angeregt, zukünftige ähnliche Forschungsvorhaben mit weiteren IoT Geräten, anderen Botnetzen und/ oder einer anderen Methodik zur Erfassung des Datenverkehrs durchzuführen.

Neben diesen Aspekten soll an dieser Stelle noch einmal auf das dargestellte Konzept und den daran anknüpfenden Implementierungsteil eingegangen werden, um einen Ausblick zu geben, wie dieses konkrete Konzept beziehungsweise der Implementierungspart weiter ausgebaut werden kann. Einer dieser Aspekte betrifft die Datenauswertung. In der Implementierung des Prototypen wurden zu Testzwecken lediglich komplette Datensätze des Datenverkehrs verarbeitet um eine ständige manuelle Wiederholung der Netzwerkaktivitäten zur Entwicklungszeit zu vermeiden. Eine Verbesserung wäre die regelmäßige Verarbeitung neu eingehender Daten des Netzwerkverkehrs um dadurch eine Echtzeitüberwachung zu ermöglichen. In Verbindung mit einer automatischen Benachrichtigung mittels der im Konzept vorgestellten Kibana Erweiterung *Alerts* wäre so bereits ein Security-Monitoring in Echtzeit möglich.

Weiterhin lässt die Entwicklung der Visualisierungskomponente viel Spielraum für mögliche Erweiterungen. Das im Konzept vorgestellte Plugin für Kibana könnte dabei so erweitert werden, dass es dem Benutzer die Möglichkeit gibt, die zu überwachenden IoT Geräte auszuwählen oder sogar automatisch zu erfassen. Für die automatische Erkennung des Gerätetyps könnte eine Betriebssystem-Erkennung zum Einsatz kommen, so dass beispielsweise Geräte mit Embedded Systemen automatisch mit in die Liste der zu überwachenden Systeme aufgenommen werden. Zudem wäre es für den Benutzer komfortabel, wenn er die Trainingsphase über die Visualisierungskomponente einleiten und nach einem gewissen Zeitraum automatisch beenden könnte um so auch zu späteren Zeitpunkten hinzugekommene Geräte mit in die Trainigsdaten aufnehmen zu können.

Abschließend lässt sich festhalten, dass das in der Arbeit vorgestellte Konzept in wesentlichen Teilen mittels der Prototypen getestet wurde, wobei sehr gute Ergebnisse erzielt werden konnten. Dennoch zeigt die Diskussion zahlreiche Ansatzpunkte für zukünftige Forschungsvorhaben auf. Diese bestehen neben der Verbesserung bereits umgesetzter Konzeptteile, auch in der Implementierung noch nicht umgesetzter Konzeptteile. Aus all diesen Erkenntnissen ergeben sich demzufolge zahlreiche Ansatzpunkte für zukünftige Forschungsvorhaben auf dem Gebiet der IoT-Sicherheit.

Literaturverzeichnis

[AIM10] Atzori, Luigi ; Iera, Antonio ; Morabito, Giacomo: The Internet of Things: A survey. In: *Computer Networks* 54 (2010), Nr. 15, S. 2787–2805. http://dx.doi.org/10.1016/j.comnet.2010.05.010. – DOI 10.1016/j.comnet.2010.05.010

[Ali09] Ali, ABM S.: Support Vector Machine: Itself an Intelligent Systems. In: *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications*. IGI Global, 2009, S. 501–522

[Ash09] Ashton, Kevin: That 'Internet of Things' Thing. In: *RFiD Journal* 22 (2009), Nr. 7, S. 97–114

[Bej04] Bejtlich, Richard: *The Tao of network security monitoring: beyond intrusion detection*. Pearson Education, 2004

[BL94] Barnett, V ; Lewis, T: *Outliers in Statistical Data (Probability & Mathematical Statistics)*. 1994

[Ble17] Bleich, Holger: *DDoS-Attacke legt Twitter, Netflix, Paypal, Spotify und andere Dienste lahm*. heise online. https://www.heise.de/newsticker/meldung/DDoS-Attacke-legt-Twitter-Netflix-Paypal-Spotify-und-andere-Dienste-lahm-3357289.html. Version: Oktober 2017. – [Online; Zugriff am 28.03.2017]

[BSI17] BSI: *IT Grundschutz Kataloge Glossar*. https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar_node.html. Version: 2017. – [Online; Zugriff am 28.03.2017]

[Cat11] Cattell, Rick: Scalable SQL and NoSQL data stores. In: *Acm Sigmod Record* 39 (2011), Nr. 4, S. 12–27

[CLR10] Chui, Michael ; Löffler, Markus ; Roberts, Roger: The internet of things. In: *McKinsey Quarterly* 2 (2010), Nr. 2010, S. 1–9

[CMM83] Carbonell, Jaime G. ; Michalski, Ryszard S. ; Mitchell, Tom M.: An overview of machine learning. In: *Machine learning*. Springer, 1983, S. 3–23

[Das99] Dasgupta, Dipankar: Immunity-based intrusion detection system: a general framework. In: *Proc. of the 22nd NISSC* Bd. 1, 1999, S. 147–160

[FGK+10] Färber, Ines ; Günnemann, Stephan ; Kriegel, Hans-Peter ; Kröger, Peer ; Müller, Emmanuel ; Schubert, Erich ; Seidl, Thomas ; Zimek, Arthur: On using class-labels in evaluation of clusterings. In: *MultiClust: 1st international workshop on discovering, summarizing and using multiple clusterings held in conjunction with KDD*, 2010, S. 1

[FPSS96a] Fayyad, Usama ; Piatetsky-Shapiro, Gregory ; Smyth, Padhraic: From data mining to knowledge discovery in databases. In: *AI magazine* 17 (1996), Nr. 3, S. 37

[FPSS96b] Fayyad, Usama ; Piatetsky-Shapiro, Gregory ; Smyth, Padhraic: The KDD process for extracting useful knowledge from volumes of data. In: *Communications of the ACM* 39 (1996), Nr. 11, S. 27–34

[FSME15] Fiedler, Roman ; Skopik, Florian ; Mandl, Thomas ; Einzinger, Kurt: Erkennen von Anomalien und Angriffsmustern. In: *Cyber Attack Information System* (2015), Januar. http://dx.doi.org/10.1007/978-3-662-44306-4_5. – DOI 10.1007/978–3–662–44306–4_5

[FSR09] Feily, Maryam ; Shahrestani, Alireza ; Ramadass, Sureswaran: A survey of botnet and botnet detection. In: *Emerging Security Information, Systems and Technologies, 2009. SECURWARE'09. Third International Conference on* IEEE, 2009, S. 268–273

[FT02] Fielding, Roy T. ; Taylor, Richard N.: Principled design of the modern Web architecture. In: *ACM Trans. Internet Techn.* 2 (2002), Nr. 2, S. 115–150. http://dx.doi.org/10.1145/514183.514185. – DOI 10.1145/514183.514185

[GBMP13] Gubbi, Jayavardhana ; Buyya, Rajkumar ; Marusic, Slaven ; Palaniswami, Marimuthu: Internet of Things (IoT): A vision, architectural elements, and future directions. In: *Future Generation Computer Systems* 29 (2013), Nr. 7, S. 1645–1660

[HELD11] Han, Jing ; E, Haihong ; Le, Guan ; Du, Jian: Survey on NoSQL database. In: *Proc. 6th Int. Conf. Pervasive Computing and Applications*, 2011, S. 363–366

[IBM09] IBM: IBM 2009 Annual Report. In: *IBM 2009 Annual Report* (2009)

[Jad15] Jadoul, Marc: *The IoT: The next step in internet evolution.* https://insight.nokia.com/iot-next-step-internet-evolution. Version: März 2015. – [Online; Zugriff am: 15.03.2017]

[JVW+14] Jing, Qi ; Vasilakos, Athanasios V. ; Wan, Jiafu ; Lu, Jingwei ; Qiu, Dechao: Security of the Internet of Things: perspectives and challenges. In: *Wireless Networks* 20 (2014), Nr. 8, S. 2481–2501

[KKT14] Keoh, Sye L. ; Kumar, Sandeep S. ; Tschofenig, Hannes: Securing the internet of things: A standardization perspective. In: *IEEE Internet of Things Journal* 1 (2014), Nr. 3, S. 265–275

[LDXZ15] Li, Shancang ; Da Xu, Li ; Zhao, Shanshan: The internet of things: a survey. In: *Information Systems Frontiers* 17 (2015), Nr. 2, S. 243–259

[LJZ09] Li, Chao ; Jiang, Wei ; Zou, Xin: Botnet: Survey and case study. In: *innovative computing, information and control (icicic), 2009 fourth international conference on* IEEE, 2009, S. 1184–1187

[Luz13] Luz, Pedro M.: *Botnet Detection Using Passive DNS*, Master Thesis/Pedro Marques da Luz, Diss., 2013

[May09] Mayer, Christoph P.: Security and privacy challenges in the internet of things. In: *Electronic Communications of the EASST* 17 (2009)

[MF10] Mattern, Friedemann ; Floerkemeier, Christian: From the Internet of Computers to the Internet of Things. In: *From active data management to event-based systems and more.* Springer, 2010, S. 242–259

[MST94] Michie, Donald ; Spiegelhalter, David J. ; Taylor, Charles C.: *Machine learning, neural and statistical classification.* Citeseer, 1994

[Nob06] Noble, William S.: What is a support vector machine? In: *Nature biotechnology* 24 (2006), Nr. 12, S. 1565–1567

[PA10] Partida, Alberto ; Andina, Diego: *IT Security Management: IT Securiteers-Setting Up an IT Security Function.* Bd. 61. Springer Science & Business Media, 2010

[PVG+11] Pedregosa, F. ; Varoquaux, G. ; Gramfort, A. ; Michel, V. ; Thirion, B. ; Grisel, O. ; Blondel, M. ; Prettenhofer, P. ; Weiss, R. ; Dubourg, V. ; Vanderplas, J. ;

Passos, A. ; Cournapeau, D. ; Brucher, M. ; Perrot, M. ; Duchesnay, E.: Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830

[Raz13] Raza, Shahid: *Lightweight security solutions for the internet of things*, Mälardalen University, Västerås, Sweden, Diss., 2013

[RNL11] Roman, Rodrigo ; Najera, Pablo ; Lopez, Javier: Securing the internet of things. In: *Computer* 44 (2011), Nr. 9, S. 51–58

[RWV13] Raza, Shahid ; Wallgren, Linus ; Voigt, Thiemo: SVELTE: Real-time intrusion detection in the Internet of Things. In: *Ad hoc networks* 11 (2013), Nr. 8, S. 2661–2674

[SBK16] Schaumüller-Bichl, Ingrid ; Kolberger, Andrea: IoT als Herausforderung für die Informationssicherheit. In: *e & i Elektrotechnik und Informationstechnik* 133 (2016), Nr. 7, S. 319–323

[Sci17] Scikit-learn, Documentation: *One-Class SVM Novelty Detection.* http://scikit-learn.org/stable/_images/sphx_glr_plot_oneclass_001.png. Version: 2017. – [Online; Zugriff am 25.03.2017]

[SSPS13] Silva, Sérgio SC ; Silva, Rodrigo M. ; Pinto, Raquel C. ; Salles, Ronaldo M.: Botnets: A survey. In: *Computer Networks* 57 (2013), Nr. 2, S. 378–403

[Sta17] Statista, Statistik Portal: *Prognose zur Anzahl der vernetzten Geräte im Internet der Dinge (IoT) weltweit in den Jahren 2014 bis 2020.* Statista Statistik Portal. https://de.statista.com/statistik/daten/studie/537093/umfrage/anzahl-der-vernetzten-geraete-im-internet-der-dinge-iot-weltweit/. Version: März 2017. – [Online; Zugriff am 21.03.2017]

[SU16] Stout, William M. ; Urias, Vincent E.: Challenges to securing the Internet of Things. In: *Security Technology (ICCST), 2016 IEEE International Carnahan Conference on* IEEE, 2016, S. 1–8

[SWZL12] Suo, H. ; Wan, J. ; Zou, C. ; Liu, J.: Security in the Internet of Things: A Review. In: *Proc. Int. Conf. Computer Science and Electronics Engineering* Bd. 3, 2012, S. 648–651

[TCG+15] Tian, C. ; Chen, X. ; Guo, D. ; Sun, J. ; Liu, L. ; Hong, J.: Analysis and design of security in Internet of things. In: *Proc. 8th Int. Conf. Biomedical Engineering and Informatics (BMEI)*, 2015, S. 678–684

[TW10] Tan, Lu ; Wang, Neng: Future internet: The internet of things. In: *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)* Bd. 5 IEEE, 2010, S. 5–376

[Vla13] Vlasveld, Roemer: *Introduction to One-class Support Vector Machines.* http://rvlasveld.github.io/blog/2013/07/12/introduction-to-one-class-support-vector-machines/. Version: Juli 2013. – [Online; Zugriff am 22.03.2017]

[WAX15] Whitmore, Andrew ; Agarwal, Anurag ; Xu, Li: The Internet of Things - A survey of topics and trends. In: *Information Systems Frontiers* 17 (2015), April, Nr. 2, 261. http://dx.doi.org/10.1007/s10796-014-9489-2. – DOI 10.1007/s10796–014–9489–2. – ISSN 1572–9419

[Wik17] Wikipedia, The free encyclopedia: *Wikipedia - Network packet.* https://en.wikipedia.org/wiki/Network_packet. Version: März 2017. – [Online; Zugriff am 25.03.2017]

[WLZH11] Winter, Philipp ; Lampesberger, Harald ; Zeilinger, Markus ; Hermann, Eckehard: Anomalieerkennung in Computernetzen. In: *Datenschutz und Datensicherheit-DuD* 35 (2011), Nr. 4, S. 235–239

[WWM04] Wang, Yanxin ; Wong, Johnny ; Miner, Andrew: Anomaly intrusion detection using one class SVM. In: *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC* IEEE, 2004, S. 358–364

[Yu09] Yu, Hwanjo: Support Vector Machine. In: *Encyclopedia of Database Systems* (2009), S. 2890–2892

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

Hamburg, 6. April 2017 Tim Eckhardt