HOCHSCHULE FÜR
TECHNIK UND WIRTSCHAFT
DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Fakultät Informatik/Mathematik

Bachelorarbeit

zum Thema

Verbreitung und Funktionsweise von Botnetzen

vorgelegt von Philipp Wenskus
im Studiengang Informatik
Matrikelnummer 37869

Betreuer Prof. Dr.-Ing. Jörg Vogt
Prof. Dr.-Ing. Robert Baumgartl

Tag der Einreichung 19.06.2017

Inhaltsverzeichnis

Kapitel 1
Einleitung

Inhalt des Kapitels

"Botnet is like sleepwalking. You have a respectable day job, a nice life, and you don't even realize you're doing it. But by night you're wandering the streets and relieving yourself on your neighbor's lawns."

Nick Asbury

1.1 Motivation

Am 21.10.2016 wurden die Server von Dyn, ein Internetdienstleister, welcher einen Großteil der Domain Name System (DNS) Infrastruktur kontrolliert, Opfer eines Hackerangriffs. Dabei handelte es sich um einen Distributed Denial of Service (DDoS) Angriff, bei dem die Server mit Anfragen bzw. Traffic überhäuft werden, bis sie unter der Last zusammenbrechen [1]. Eine Folge dessen war, dass unter anderem Seiten von großen Konzernen wie Amazon, Netflix, Twitter und GitHub, alles Kunden von Dyn, zeitweise nicht mehr zu erreichen waren [2, 3].
Hinter der Attacke stand, wie bei DDoS-Angriffen üblich, ein oder mehrere Botnetze, d. h. ein Netzwerk aus kompromittierten PCs. Mit Berichten von ca. 100 000 infizierten Rechnern und einer Angriffs-Stärke von bis zu 1,2 Terrabit pro Sekunde(Tbps) handelte es sich hierbei um die mit Abstand größte Attacke ihrer Art [3, 4]. Hauptverantwortlich für die Attacke war dabei das Mirai-Botnetz [4], ein Netz, welches aus unsicheren Internet of Things (IoT) Geräten besteht. Kurz vor dem Angriff wurde der Code dieses Botnetzes in einem Hacker-Forum veröffentlicht [5].

1.2 Zielsetzung und Aufbau der Arbeit

Das Ziel der Arbeit soll sein, den Lesern einen Überblick über die Verbreitung und Funktionsweisen von modernen Botnetzen zu verschaffen.

Dazu wird im ersten Teil der Stand der Technik behandelt. In diesem zweiten Kapitel wird anfangs der Begriff Botnetz definiert und ein Einblick in die Entstehung der Botnetze gegeben. Danach werden die Grundkomponenten und der normale Lebenszyklus erklärt. Anschließend wird diskutiert wie sich die modernen Botnetze verbreiten und was deren Angriffstechniken sowie deren Ziele sind. Zum Schluss wird ein Überblick über die grundlegenden Erkennungstechniken von Botnetzen gegeben.

Im zweiten Teil wird das Mirai-Botnetz, welches für die unter 1.1 beschriebene DDoS Attacke auf Dyn hauptverantwortlich war, genauer beleuchtet. Dazu wird der veröffentlichte Programmcode des Netzes analysiert. Zudem wird die Architektur des Botnetzes sowie der angepasste Lebenszyklus des Mirai-Bots dargestellt. Danach werden die Attacken, die in diesem Netz zur Verfügung stehen, beschrieben und abschließend Daten, die das Infektionsrisiko zeigen sollen, ausgewertet.

Zum Schluss wird diskutiert, wer für die ausreichende Sicherung der Geräte verantwortlich ist. Es wird auch ein Ausblick in die Zukunft gegeben, wie sich die Botnetze in den nächsten Jahren entwickeln könnten.

Kapitel 2

Stand der Technik

Inhalt des Kapitels

In diesem Kapitel sollen die verschiedenen von Botnetzen genutzten Techniken erklärt und veranschaulicht werden.

2.1 Definitionen

2.1.1 Malware

Malware (dt. Schadsoftware), die Kurzform der englischen Wörter "malicious software", bezeichnet Computerprogramme, welche zum Ausführen von bösartigen bzw. schädlichen Funktionen genutzt werden [6–8]. Dabei handelt es sich um einen Oberbegriff, der verschieden klassifiziert werden kann, um die unterschiedlichen Arten der Malware einzuteilen [7]. Eine grobe Einteilung der Schadsoftware wäre u.A. Würmer und Trojaner, wobei Trojaner laut dem Pandaslab Quartals Report aus dem ersten Quartal 2016 mit $65,89\,\%$ aktuell die meistverbreitete Malware-Art ist(siehe auch Abbildung 2.1.1) [6, 8, 9]. Auch der Virus, ein Name, der im allgemeinen Sprachgebrauch oft fälschlicherweise für jegliche Schadsoftware verwendet wird, ist eine dieser

Unterarten [8]. Eine genauere Aufteilung für Malware liefert z. B. der Hersteller für Sicherheitssoftware Kaspersky Lab, indem die verschiedenen Unterarten noch nach Funktionen bzw. Funktionsweisen aufgeteilt werden [7]. So kann beispielsweise der klassische Computerwurm noch in die verschiedenen Kommunikationsverfahren Instant-Messaging, Peer-to-Peer oder IRC aufgeteilt werden.

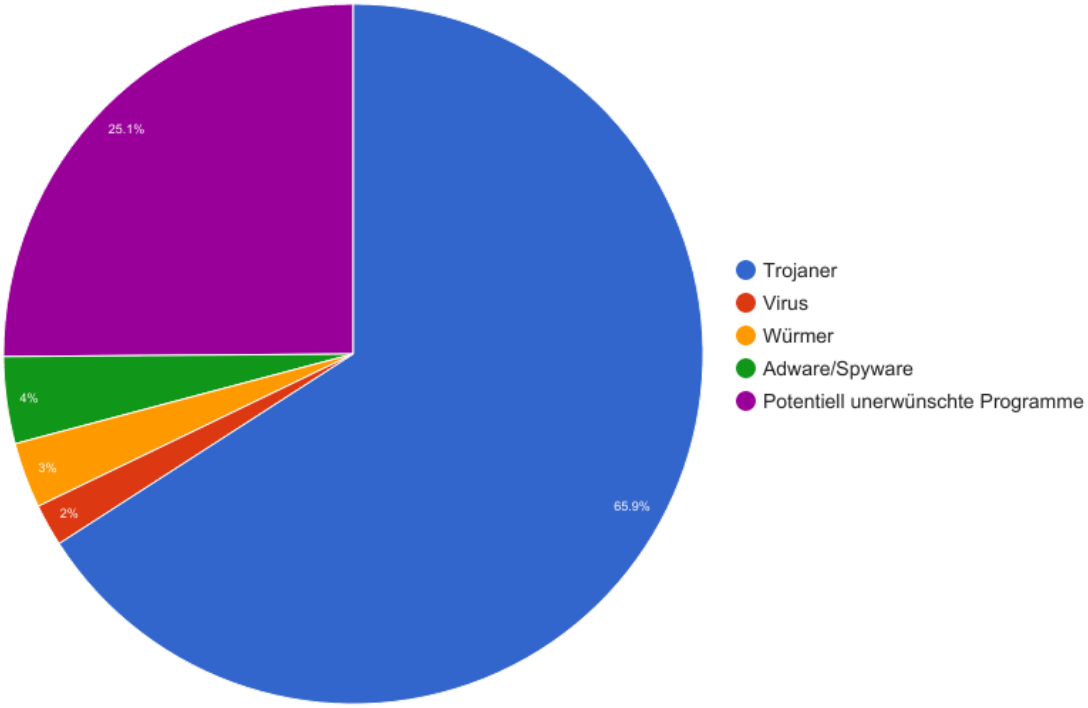

Abbildung 2.1.1: Infektionen nach Art der Malware im ersten Quartal 2016 [9]

2.1.2 Botnetze

"Botnets are like a zombie army, full of dead-eyed, half-crazed people obeying the instructions of a remote and unseen master, without realizing they are part of a destructive tribe."

Nick Asbury

Ein Botnetz ist ein aus mehreren durch Malware infizierten bzw. „versklavten“ Computern bestehendes Netzwerk, das von dem Angreifer, dem sogenannten Botmaster (in seltenen Fällen auch „botherder“ genannt), gesteuert werden kann [10].

Ein Bot (aus dem englischen "robot") ist dabei die ausführbare Datei, also die Schadsoftware welche auf dem zu infizierenden Rechner ausgeführt werden soll [11]. Im Sprachgebrauch werden auch die einzelnen teilnehmenden Computer oft Bot oder Zombie genannt [12]. Es ist schwer, die Schadsoftware der einzelnen Bot-Rechner in eine der unter Abschnitt 2.1.1 genannten Kategorien einzuteilen, da sie sich mittlerweile „ausbreiten wie Würmer, vor der Erkennung verstecken wie Viren und Angriffe wie viele Stand-alone-Tools ausführen können “ [13]. Die Benutzer der Computer sind dabei meistens ahnungslos und merken nicht, dass das System infiziert wurde. Laut einem Bericht vom Bundesamt für Sicherheit in der Informationstechnik (BSI) aus dem Jahr 2016 wurden mithilfe der Internet Service Provider (ISP) täglich bis zu 39 000 Infektion deutscher Systeme durch Bots registriert [14]. Einigen Experten zufolge sind 16-25% aller mit dem Internet verbundenen Geräte Teil eines solchen Botnetzes [10].

2.2 Geschichte der Botnetze

Ursprünglich wurden Bots für das Internet Relay Chat (IRC), ein textbasiertes Chat-Protokoll entwickelt. Dabei ging es aber nicht darum Teilnehmern der Chats Schaden zuzufügen, sondern die Interaktionen in den Chaträumen zu kontrollieren. Einer der ersten Chatbots, GM(Game Manager), wurde dabei 1989 von Greg Lindahl erstellt. GM erlaubte es den IRC Usern "Hunt the Wumpus", ein textbasiertes Spiel, im Chat zu spielen. 1993 wurde dann der von Jeff Fisher erstellte Eggdrop-Bot veröffentlicht. Dieser bis heute weiterentwickelte Bot bietet verschiedene Funktionen zur Unterstützung der IRC-Channel-Verwaltung [10,11].

Im Jahr 1998 tauchte dann der erste bösartige Bot auf, der GT (Global Threat) Bot, der es ermöglichte einen versteckten mIRC-Client (ein Instant-Messenger-Client, der das IRC Protokoll unterstützt) auf dem Zielrechner auszuführen. So konnte der GT-Bot über IRC-Befehle z. B. eine Denial of Service (DoS) Attacke instruieren [15]. Ab diesem Zeitpunkt wurden immer mehr bösartige Bots gefunden. Durch die Verwendung von anderen Protokollen, wie Hypertext Transfer Protocol (HTTP) oder Peer-to-Peer (P2P) Protokollen, wurde die Kommunikation zwischen Botmaster und Bot weiterentwickelt. Auch die Netz-Architekturen der aus mehreren Bots entstehenden Botnetze wurden ausgebaut, ebenso wie die Angriffsmethoden, die

sich beispielsweise von einfachen DoS Attacken zu komplexeren DDoS Angriffen, bei dem mehrere Angreifer ein Ziel angreifen, verbesserten, um die Botnetze raffinierter und widerstandsfähiger zu machen [10].

2.3 Komponenten eines Botnetzes

Da Botnetze sehr unterschiedlich aufgebaut sind, werden in diesem Abschnitt nur die Kernkomponenten behandelt. Diese, nämlich die C&C-Struktur sowie der/die Bot/Bots, kommen in der Regel in jedem Botnetz vor. Es gibt zwar oft mehr Komponenten, jedoch sind nur die beiden hier genannten essentiell für die Funktionsfähigkeit.

Command and Control (C&C) Struktur

Die meisten Botnetze bieten dem Botmaster die Möglichkeit Befehle an die Zombie-Rechner zu senden. Dazu wird eine sogenannte Command and Control (C&C) Struktur eingerichtet. Meistens wird für die Umsetzung einer solchen Struktur ein C&C Server verwendet, mit dem die infizierten Computer eine Verbindung aufbauen und auf Befehle warten. So kann der Botmaster das gesamte Netz steuern. Je nach Architektur kann ein Botnetz aber auch ohne C&C Server auskommen (siehe Abschnitt 2.5) [10].

Die Herausforderungen, die auf einen Angreifer zukommen sind die Folgenden [13]:

1. Wie kann ich möglichst schnell alle infizierten Computer erreichen?
2. Wie kann ich die Kommunikation verschleiern, um unerkannt zu bleiben?
3. Wie kann ich bei Entdeckung verhindern, dass die Herkunft des Befehls, also die Adresse des C&C erkannt wird?

Bot

Der Bot ist nicht nur der Namensgeber des Netzwerkes, sondern auch fester Bestandteil von diesem. Er ist die Schadsoftware, die auf einem verwundbaren Rechner installiert und ausgeführt wird. Dabei gibt es, wie in Abschnitt 2.6 nachzulesen,

mehrere Methoden wie dieser ins System des Opfers gelangt. Ist die Software einmal installiert startet sie normalerweise nach jedem Systemstart automatisch neu und bleibt so bis zu ihrer Entdeckung, z. B. durch ein Antivirenprogramm, für den Botmaster verfügbar [10]. Ist der Rechner infiziert wird der ganze Computer als Bot oder auch Zombie bezeichnet.

2.4 Lebenszyklus eines Bots

Hier soll ein typischer Lebenszyklus eines Bots nach Jelasity und Bilicki [16] und Silva et al. [10] dargestellt werden. Wichtig ist, dass dieser allgemein gehalten ist und nicht zwingend bei jedem Botnetz genau so abläuft. Der Verlauf ist in Abbildung 2.4.1 graphisch dargestellt. Analog dazu werden im Folgenden die einzelnen Phasen erklärt:

1. **Infektionsphase:** In der ersten Phase wird in den Rechner eingedrungen. Dies geschieht beispielsweise über das Ausnutzen einer Sicherheitslücke. Mehr dazu kann im Abschnitt 2.6 gefunden werden.

2. **Injektionsphase:** Wurde sich erfolgreich Zugriff zum System verschafft, wird nun in der zweiten Phase versucht die Binary-Datei des Bots, also die eigentliche Schadsoftware herunterzuladen und zu installieren. Nach der Installation zählt der Computer als Zombie-Rechner. Die Injektionsphase wird in der Literatur oft mit der Infektionsphase zusammengefasst [10].

3. **Verbindungsphase:** Nach der Installation stellt der Computer in der dritten Phase eine Verbindung mit dem C&C Server her.

4. **Bösartige Aktivitäten:** In der Phase für bösartige Aktivitäten wird auf Befehle des Botmasters gewartet, um Angriffe auszuführen. Die Angriffe könnten z. B. DDoS oder Spam-Attacken sein. Mehr dazu kann unter Abschnitt 2.7 nachgelesen werden. An dieser Stelle kann die Verbindungsphase noch einmal wiederholt werden, da es vorkommen kann, dass sich der Zombie-Rechner immer wieder beim C&C Server zurückmelden muss, z. B. nach einem Neustart des Systems.

5. **Wartung und Upgrade:** In der letzten Phase geht es um das Instandhalten und Upgraden des Bots. Hier werden neue Features hinzugefügt, Programmfehler entfernt oder sogar ein neuer C&C Server eingestellt. Nachdem die Bots auf dem aktuellen Stand sind verbinden sie sich erneut mit dem C&C Server.

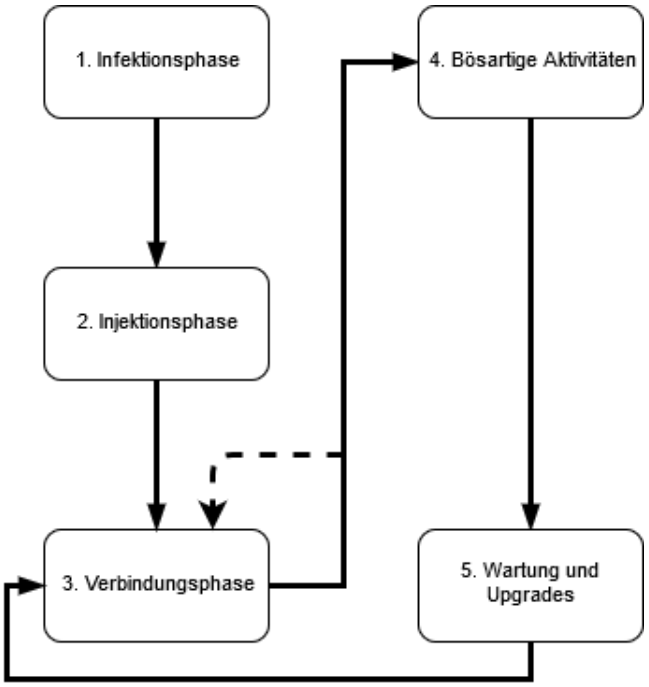

Abbildung 2.4.1: Allgemeiner Lebenszyklus eines Botnetzes nach Silva et al. [10]: Zu sehen sind die 5 Phasen, die ein Bot durchläuft. Der gestrichelte Pfeil von 4. zu 3. deutet an, dass die Verbindungsphase wiederholt werden kann, wenn der Bot sich immer wieder beim C&C Server zurückmelden muss. Auch nach einem Upgrade (5. Phase) wird die Verbindungsphase erneut durchgeführt.

2.5 Verschiedene Architekturen

Hier wird erklärt wie Botnetze aufgebaut, das heißt miteinander verbunden sind bzw. kommunizieren. Dabei unterscheidet man in der Regel zwischen einer zentralisierten und einer dezentralisierten C&C-Infrastruktur. Darüber hinaus gibt es Mischformen der beiden Architekturen [10, 13].

2.5.1 Zentralisierte Architektur

Bei der zentralisierten Architektur verbinden sich die Zombie-Rechner nach ihrer Infektion mit einem, wie der Name schon vermuten lässt, zentralen Punkt. Dieser Punkt ist dabei in der Regel der C&C-Server, also der Server, der für das Senden von Befehlen an die Bots verantwortlich ist. Die verschiedenen zentralisierten Topologien werden im Folgenden aufgelistet. Außerdem werden danach kurz die gängigsten Kommunikationsprotokolle für zentralisierte Architekturen beschrieben und auf die Verschleierungstechniken des C&C Servers eingegangen.

Sternstruktur

Die Sternstruktur ist das einfachste zentralisiert aufgebaute Netzwerk. Wie in Abbildung 2.5.1 zu sehen ist, verbinden sich alle Zombies direkt mit dem C&C Server. Dies bringt den Vorteil einer schnellen Kommunikation mit den Bots, jedoch auch einen entscheidenden Nachteil: Da sich alle Bots direkt mit dem C&C Server verbinden, ist dieser sehr leicht aufzuspüren. Wird der Server nun angegriffen, übernommen oder abgeschaltet führt dies zum Kontrollverlust über das gesamten Netzwerk. Der C&C Server ist somit der „Flaschenhals“ oder auch der sogenannte Single Point of Failure (SPOF) der Infrastruktur, denn sollte dieser ausfallen, fällt auch das gesamte Botnetz aus. [17]

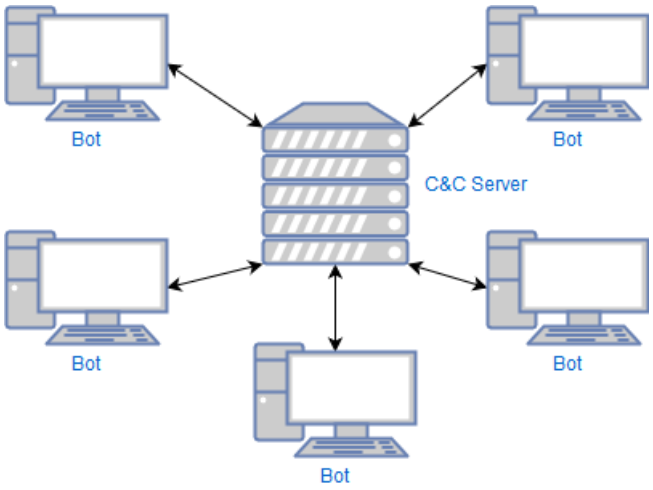

Abbildung 2.5.1: Botnetz mit Sternstruktur: Jeder Bot verbindet sich direkt mit dem C&C Server.

Multi-Server Struktur

Die Multi-Server Struktur unterscheidet sich, wie in Abbildung 2.5.2 zu sehen ist, von der Sternstruktur im Wesentlichen durch den Einsatz von mehreren C&C Servern. Hier wird so vorgegangen, dass die Bots die verschiedenen Adressen der Server kennen und sich mit einem beliebigen verbinden. Der höhere Planungsaufwand, der dadurch entsteht wird mit dem Wegfallen des unter Abschnitt 2.5.1 genannten Single Point of Failure(SPOF) belohnt. Denn wenn ein Server ausgefallen ist, verbindet sich der Zombie-Rechner einfach mit einem anderen und der Angreifer verliert somit nicht sofort die Kontrolle über das Botnetz [17].

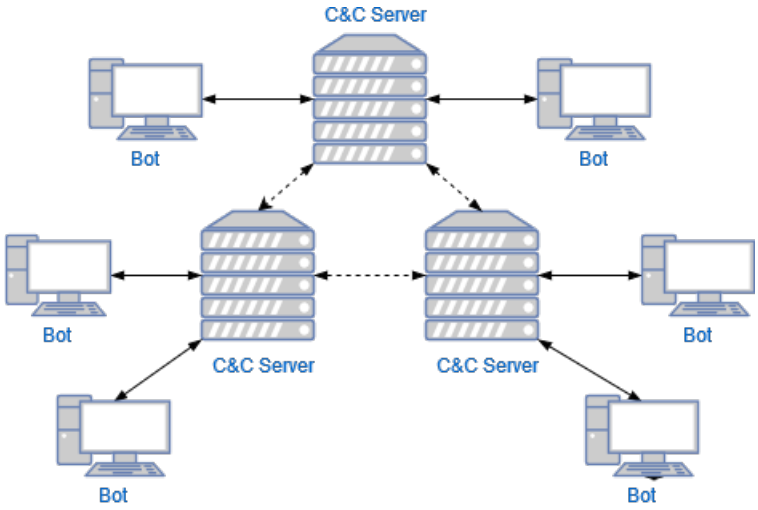

Abbildung 2.5.2: Botnetz mit Multi-Server Struktur: Es gibt mehrere C&C Server, mit denen sich die Zombie-Rechner verbinden können.

Hierarchische Struktur

Eine hierarchische Struktur des Netzwerks erweitert die Multi-Server um eine Proxyschicht zwischen C&C Server(n) und den Bot-Rechnern. Wenn ein infizierter Computer nun eine Anfrage stellt, wird diese zum Proxy Server geschickt, der die Nachricht dann an den C&C Server weiterleitet. Umgekehrt funktioniert das für die Befehle des C&C Servers genauso. Entscheidender Vorteil dieser Methode ist, dass diesmal nur die Adresse des jeweiligen Proxys bekannt wird und nicht die des eigentlichen C&C Servers. Um einen ausgefallenen Server ersetzen zu können, werden hier oft Domains statt hart-codierter IP-Adressen verwendet. Somit können

die beim DNS-Anbieter eingestellten DNS-Records bei Ausfall einfach an eine neue Serveradresse angepasst werden [17]. Der Aufbau einer hierarchischen Struktur ist in der Abbildung 2.5.3 dargestellt.

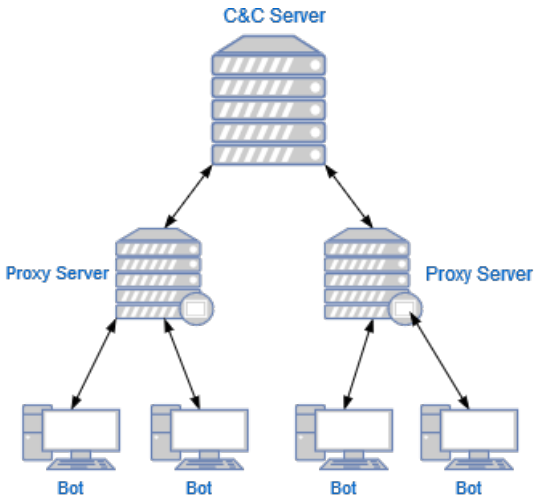

Abbildung 2.5.3: Ausschnitt eines Botnetzes mit hierarchischer Struktur: Zwischen infizierten Computern und den C&C Server(n) liegt eine Proxyschicht, die die Kommunikation übermittelt.

2.5.1.1 Kommunikationsprotokolle

In diesem Abschnitt werden kurz die unterschiedlichen Kommunikationsprotokolle bei zentralisierten Botnetzen beschrieben. Dazu wird vorallem auf die beiden hauptsächlich benutzten Protokolle, nämlich IRC und HTTP eingegangen [10]. Danach werden weitere Methoden genannt.

Internet Relay Chat (IRC)

Das erste Protokoll, das jemals für die Kommunikation in Botnetzen verwendet wurde, ist IRC, ein Chatprotokoll. Die Zombie-Rechner in solchen Botnetzen verbinden sich mit einem IRC-Server, loggen sich auf einem zuvor festgelegten Channel ein und warten dort auf Befehle ihres Masters. Wenn dieser seine Zombies befehligen will, muss er sich nur noch in den festgelegten Chat einwählen und kann dort seine Befehle versenden. Diese werden von den Bots gelesen, interpretiert und ausgeführt. Der Vorteil dieser Kommunikation ist vor allem die einfache Umsetzung, da Bibliotheken für das Protokoll fertig bereitstehen. Jedoch hat sie auch einen entscheidenden

Nachteil: Sollte der IRC-Server abgeschaltet werden, verliert der Botmaster die Kontrolle über das gesamte Botnetz. Es entsteht also ein SPOF wie auch bei der Sternstruktur [17].

Hypertext Transfer Protocol (HTTP)

Das HTTP ist ein immer beliebter werdendes Protokoll für die Kontrolle von Botnetzen. Über eine vordefinierte Adresse verbinden sich die Zombies mit dem Webserver und warten dort auf neue Befehle. Die zunehmende Beliebtheit des Protokolls resultiert vor allem aus der einfachen Installation, der leichten Zugänglichkeit zum Webserver sowie der Möglichkeit, das Bedieninterface für das Botnetz direkt auf dem Webserver zu installieren. Aber auch hier entsteht ein SPOF. Sollte der Webserver abgeschaltet werden, kann das Botnetz nicht mehr befehligt werden [17]. Abhilfe würde hier der Einsatz von mehreren Webservern schaffen, wie es bei der Multi-Server-Architektur der Fall ist.

Weitere Kommunikationsmöglichkeiten

Es gibt noch eine Reihe weiterer Kommunikationsmöglichkeiten, welche aber selten genutzt werden. So ist es möglich, die Kommunikation über Instant Messaging Dienste wie Whatsapp oder Skype abzuwickeln oder sogar soziale Netzwerke wie Twitter oder Facebook für die Kommunikation zu missbrauchen. Eine weitere, zunehmend beliebtere Form der Informationsübermittlung ist mittels cloudbasierter Dienste wie etwa „Amazons Elastic Compute Cloud“ (Amazon EC2) oder „Google App-Engine“ [17].

2.5.1.2 Verschleierungstechnik

Bei den zentralen Architekturen kommen häufig Verfahren zum Einsatz, die verhindern sollen, dass die Adressen der zentralen Punkte des Botnetzes herausgefunden werden um somit den weiter oben oft beschriebenen Single Point of Failure (SPOF) zu umgehen. Um dies zu bewerkstelligen, wird heutzutage davon abgesehen, die IP-Adresse in den Source Code des Bots fest einzuprogrammieren. Stattdessen wird auf eine Verschleierung durch einen Domain-Namen gesetzt. Dieser alleine bringt schon den Vorteil mit sich, dass man bei Bedarf einfach die Adresse auf die die Domain verweist, ändern kann. Somit lässt sich z. B. ein C&C Server einfach

ersetzen. Oftmals kommen sogar mehrere Domains zum Einsatz, von denen der Bot eine auswählt. Hier bleibt jedoch das Problem des SPOF bestehen, da durch eine Abschaltung der Domain(s) der C&C Server nicht mehr zu erreichen ist.
Um diese Schwachstelle zu vermeiden, setzen die Angreifer oft auf sogenannte „bulletproof“(dt. kugelsichere) DNS-Anbieter. Diese geben eine Garantie darüber, dass die Domain, selbst bei Beschwerden über Missbrauch, nicht abgeschaltet wird.

Um zu verhindern, dass die eigentliche Adresse des C&C Servers bekannt wird, wird bei Botnetzen deshalb immer öfter auf ein Fast-Flux Service-Netzwerk zurückgegriffen. Dazu wird ein DNS-System benutzt, welches über ständig wechselnde Proxy-Knoten den Netzwerkverkehr zwischen C&C Server und Bot-Rechner weiterleitet. Es wird ein Domainname festgelegt, der einen Teil der IP-Adressen der verfügbaren Proxy Knoten auflöst, wobei diese Namenseinträge eine sehr kurze Lebensdauer (Time-to-Live, TTL) haben, und oftmals im Abstand von wenigen Minuten in andere Proxy-Knoten-IPs geändert werden. Als Proxy-Knoten werden dabei gerne Bots verwendet, die öffentlich erreichbar sind und eine gute Internetanbindung besitzen. Das Fast-Flux-Verfahren verringert durch die Verwendung ständig wechselnder Proxy-Knoten stark das Risiko, dass der C&C Server gefunden und das Botnetz ausgeschaltet werden kann. Um nicht nur von einer einzigen Domain abhängig zu sein, wird hier neben den „bulletproof“ DNS-Anbietern ein sogenannter Domain-Generierungs-Algorithmus (DGA), benutzt, welcher in die Bots einprogrammiert wird. Der Zombie-Rechner berechnet darüber, in Abhängigkeit eines bestimmten Wertes, eine Liste von möglichen Domain-Namen und versucht sich dann mit einem dieser Namen zu verbinden. Schlägt dies fehl, wird die nächste Adresse verwendet. Um sicherzustellen, dass auch alle Bots zur selben Zeit die selbe Liste generieren wird als Rechengrundlage z. B. das aktuelle Datum benutzt. Da der Botmaster weiß, welche Domains der DGA gerade berechnet, kann er sich eine der Domains aus der Liste aussuchen, diese kurzzeitig registrieren und darüber dann neue Befehle an die Bots senden. Somit ist das Botnetz unabhängig von dem Domain-Namen und eine Abschaltung von diesem ist noch schwerer zu bewirken [18, p. 56-57] [19].

2.5.2 Dezentralisierte Botnetze

Neben den zuvor beschriebenen zentralen Botnetzen gibt es auch dezentralisierte Botnetze. Im Gegensatz zu zentralisierten C&C Infrastrukturen bieten diese mehr Flexibilität und Widerstandsfähigkeit. Außerdem sind die Botnetze durch das Fehlen des zentralen C&C Servers viel schwerer zu zerschlagen. Da es sich hierbei ausschließlich um P2P Netze handelt, wird in der Regel auch nur über P2P Protokolle kommuniziert.

Peer to Peer (P2P) Botnetze

Die meisten dezentralen Botnetze basieren auf einem P2P Netzwerk. Übersetzt bedeutet Peer etwa ebenbürtig oder gleichrangig, was schon das Prinzip dieser Netzwerke erklärt, nämlich, dass alle Teilnehmer gleichberechtigt sind. Das bedeutet, es gibt keinen Server, der die Bots kontrolliert, sondern jeder Bot stellt einen Netzwerknoten dar, indem er gleichermaßen Server wie auch Client ist. Diese Eigenschaft erschwert die Überwachung des Botnetzes erheblich und steigert außerdem seine Widerstandskraft enorm. Sollten Knoten ausfallen, werden diese einfach durch andere ersetzt. Dies klappt selbst wenn ein Großteil der Knoten abgeschaltet wird, was ein Zerschlagen des Botnetzes fast unmöglich macht. Möchte der Botmaster nun mit dem Botnetz interagieren, muss er sich nur als Peer in das Netzwerk einloggen und seinen Befehl versenden. Wichtig dabei ist eine ordentliche Verschlüsselung der Kommandos, da jeder Peer dazu berechtigt ist, diese Befehle auszuführen und so z. B. eine Übernahme des Botnetzes möglich wäre [17].
Die hier behandelten Botnetze können laut Jelasity und Bilicki [16] nach den benutzten P2P Overlays eingeteilt werden. Diese werden im Folgenden kurz erklärt:

Unstrukturiertes P2P Overlay

Bei unstrukturierten P2P Netzen verbinden sich die einzelnen Peers, also hier die einzelnen Zombie-Rechner, zufällig mit anderen Peers. Sie bieten keine Möglichkeit für Routing, d. h. es können keine festen Verbindungswege eingestellt werden. Zudem gibt es in dieser unstrukturierte Topologie kein „Key-Lookup“, was bedeutet, dass sich die Peers untereinander nicht über einen Identifikationsschlüssel erkennen können. Um in solchen Netzen zu suchen, bieten sich die Algorithmen Flooding (Breitensuche) und Random-Walk (Tiefensuche) an [16, 20].

Super-Peer Overlay

In Super-Peer Netzwerken sind nicht alle Peers gleich, was eigentlich dem Grundgedanken eines klassischen P2P Netzwerks widerspricht. Hierbei werden automatisch sogenannte Super-Peers ausgewählt, welche enger miteinander verbunden sind als die restlichen Knoten. Jeder andere Peer verbindet sich dann genau mit einem Super-Peer und gibt diesem Informationen preis. Will man nun im Netzwerk suchen, genügt es über die Super-Peers zu suchen. Diese Form des P2P Netzwerks wird z. B. von Skype oder Gnutella benutzt. Da das Netz jedoch leichter zu erkennen und nicht so widerstandsfähig ist wie die anderen Overlays ist, kann davon ausgegangen werden, dass die meisten Botnetze diese Technik nicht verwenden [16, 20].

Strukturiertes Overlay

In strukturierten P2P Netzwerken wird eine bekannte Topologie erzeugt, welche es erlaubt eine zielgerichtete Suche nach Inhalten durchzuführen. Dabei wird normalerweise auf eine Distributed Hash Table (DHT) zum Routing zurückgegriffen. Für diese Tabellen gibt es verschiedene Algorithmen wie CAN, Chord oder Kademlia, um nur ein paar davon zu nennen. Die meisten P2P Botnetze setzen heutzutage auf diese strukturierten Overlays [10, 16, 20].

2.5.3 Hybrid Architektur

Neben den oben genannten Architekturen gibt es auch Hybrid-Netzwerke, die eine Mischung aus zentralisierten und dezentralisierten Netzwerken sind. Dazu werden Bots, welche einem solchen Netzwerk angehören in sogenannte „Diener“- und Client-Bots eingeteilt. Die Diener agieren als Server und Client gleichzeitig und sind mit statischen und routbaren IP-Adressen konfiguriert. Die Client-Bots dagegen lassen keine Verbindung von außen zu und sind mit einer dynamischen oder einer nicht routbaren IP-Adresse ausgestattet. Dabei macht es keinen Unterschied ob die Clients direkten Internetzugriff haben oder hinter einer Firewall sitzen. Der Verbindungsablauf in einem solchen Netzwerk sieht dann wie folgt aus:
Die Diener-Bots warten auf einkommende Netzwerkanfragen auf einen bestimmten Port. Alle anderen Bots müssen sich nun immer wieder mit dem Diener-Bot in seiner Rolle als Server verbinden und nach neuen Befehlen des Botmasters fragen. Sollte der Client Befehle bekommen, die er zuvor noch nicht empfangen hat, werden diese

sofort an die anderen Diener-Bots weitergeleitet um den Befehl so möglichst schnell zu verbreiten. Um die Erkennung solcher Botnetze schwieriger zu machen, wird dabei auf eine symmetrische Verschlüsselung der Kommunikation gesetzt [10].

2.6 Verbreitung

Die meisten Computernutzer würden wohl nicht zustimmen, wenn man ihnen vorschlägt einen Bot auf ihrem Rechner zu installieren. Deshalb ist eine der größten Fragen eines Botmasters, wie er die Schadsoftware am besten auf den Computer des Opfers laden und ausführen kann. Es gibt dabei sehr viele Möglichkeiten wie das Problem gelöst wird [13]. Die meistgenutzte Variante ist der Scan nach Sicherheitslücken, welche entweder durch Sofware, Betriebssysteme oder manchmal sogar durch anderer Malware hervorgerufen werden. So hat beispielsweise der Bagel-Wurm Backdoors (Hintertüren) auf den infizierten Geräten hinterlassen, welche dann genutzt werden konnten [21]. Dieser Prozess der Verbreitung hat sich mit der Zeit immer weiter entwickelt. So waren es am Anfang noch einzelne Angriffsvektoren, d. h. es wurde nur nach einer bestimmten Sicherheitslücke gesucht, welche dann teilweise sogar noch manuell ausgenutzt werden musste. Mittlerweile haben Angreifer mehrere Angriffsvektoren, über die sich der Bot automatisch verbreiten und installieren kann.

Doch nicht nur das Ausnutzen von Sicherheitslücken wurde effizienter, sondern auch die Auswahl der Opfer. Wurden die Opfer anfangs noch zufällig ausgewählt wird mittlerweile oftmals eine „hitlist“, z. B. eine Liste aus verwundbaren Hosts, eine E-Mail Liste oder eine Userliste eines Sozialen Netzwerks, erstellt. Zusätzlich änderte sich der Angriffsvektor dahingehend, dass nicht mehr wie früher Low-Level Services angegriffen werden, sondern, dass Sicherheitslücken von High-Level Software wie Internetbrowser oder Web-Playern (z. B. der Flash Player) ausgenutzt werden [13]. Oftmals kommt auch Social Engineering zum Einsatz. Hier ist nicht mehr die Software, sondern der Mensch, der den Computer bedient die Schwachstelle. Dieser wird dann z. B. über eine Facebook Nachricht oder durch eine Webseite dazu gebracht auf einen Link zu klicken. Über diesen wird das Opfer auf eine bösartige Webseite weitergeleitet, welche dann nach Sicherheitslücken im System sucht. Werden Schwachstellen gefunden, werden diese ausgenutzt und der

Rechner damit Teil des Botnetzes. Auch Spammails mit schädlichem Anhang, welche oftmals direkt vom Botnetz verschickt werden, oder illegal geladene Software, die zuvor infiziert wurde, tragen zur Ausbreitung der Botnetze bei [22].

2.7 Angriffe und deren Ziele

Attacken, die von Botnetzen ausgehen, haben sich seit der Entstehung immer weiter entwickelt. War es anfangs noch eine einfache DoS Attacke wurde diese schnell zu komplexeren DDoS Angriffen ausgebaut. Auch das Sammeln von Informationen stellte sich als ein lukratives Geschäft für Botnetzbetreiber heraus. Die folgenden Abschnitte erklären die meistgenutzten Angriffe und deren Ziele [13].

Distributed Denial of Service (DDoS)

Eine DDoS-Attacke ist wohl die bekannteste Art mit einem Botnetz anzugreifen. Das Ziel des Angriffs besteht darin, das Netz bzw. den Server des Opfers so stark zu überlasten, dass die eigentlichen Aufgaben nicht mehr erfüllt werden können. Wenn der Botmaster nun eine solche Attacke auf eine Webseite anordnet, fangen die Bots an, zyklisch Anfragen an die Netzwerkadresse zu schicken. Dabei handelt es sich meistens um HTTP-Anfragen, neuere Botnetze wie das unten beschriebene Mirai-Botnetz bieten aber auch andere Möglichkeiten, z. B. das Senden von TCP-SYN-Paketen. Durch die Anfragen der vielen Bots wird die Bandbreite oder die Rechenkapazität des Opfers aufgebraucht und die Webseite kann durch die Überlastung nicht mehr, oder sehr schlecht erreichbar werden. Diese Nichterreichbarkeit kann für Online-Händler wie z. B. Amazon enorme Umsatzeinbußen bedeuten. Auch Unternehmen, deren Geschäft eigentlich nichts mit dem Internet zu tun hat, können attackiert werden, indem Onlinedienste, wie z. B. Kommunikationsplattformen nicht genutzt werden können. Je nach betroffenem System, kann dabei auch ein erheblicher Imageschaden entstehen.
Durch diese Verletzlichkeit der Opfer sehen die Botmaster eine Möglichkeit das Unternehmen zu erpressen oder diesem aus persönlichen Gründen, z. B. wegen schlechter Erfahrungen zu schaden [17, 23]. Außerdem geht der Trend dahin über, dass solche DDoS Botnetze vermietet werden, um Geld zu verdienen. Dies macht es möglich, mit den nötigen Finanzmitteln eine DDoS Attacke auszuführen, ohne große

Vorkenntnisse darüber zu besitzen. Ende 2016 gab es beispielsweise ein Angebot für ein Mirai-Botnetz. Hier konnte man sich 50 000 Bots 2 Wochen lang für einen Preis zwischen 3 000 und 4 000 Dollar mieten [24].

Spam

Das Versenden von Spam ist neben DDoS-Attacken ein oft genutztes Feature. So werden laut Experten 80 % aller Spam-Mails weltweit über durch Bots infizierte Rechner versandt. Den Zombies wird eine E-Mail-Liste sowie eine Mail-Vorlage bereitgestellt. Anhand dieser Liste werden die Mails über SMTP(Simple Mail Transfer Protocol) verschickt. Einen großen Vorteil bringt dabei die Möglichkeit auch Mails an die im infizierten Rechner eingespeicherten E-Mail Adressen zu senden und diese mit in die Adressliste aufzunehmen. Somit wächst die Anzahl der Opfer-Adressen stetig an. Auch hier werden die Botnetze, die Spam-Mails verschicken, vermietet um Geld zu verdienen. So können eine Million Spam-Mails für 150 bis 200 Dollar gekauft werden [17].

Informationsdiebstahl

Den Angreifern ist mittlerweile aufgefallen, welchen Wert persönliche Informationen haben können. Deshalb werden in den Bots Funktionen implementiert, die es erlauben Informationen vom Opfer-Rechner zu stehlen. So gelangt man etwa an Kreditkarteninformationen, Sozialversicherungsnummer, Adressen etc. Auch eingebaute KeyLogger, also Programme, die Tastatureingaben aufzeichnen, werden benutzt um Passwörter mitzuschneiden [13]. Die gewonnenen Informationen kann der Botmaster entweder verkaufen oder für sich selbst, z. B. zum Bezahlen im Internet nutzen.

Klickbetrug

Einige Onlinedienstleister bieten die Möglichkeit mit Klicks auf Werbung (Pay per Click) oder durch das Vermitteln neuer Besucher auf einer Website (Pay per Visit) über einen sogenannten „Reflink“ Geld zu verdienen. Diese Geschäftsmodelle werden gerne von Botmastern ausgenutzt um Geld zu generieren. Die infizierten Rechner bekommen dazu den Befehl ein automatisiertes Script auszuführen, welches auf die Werbebanner klickt oder die Seite besucht. Was hier besonders hilfreich ist, ist die

Tatsache, dass die Betreiber solcher Werbeplattformen durch die ständig wechselnden IP-Adressen und Computerinformationen der verschiedenen Bot-Rechner den Betrug nicht erkennen können [17, 23].

Weitere Angriffsmethoden

Es gibt noch zahlreiche weitere Angriffsmöglichkeiten für Botnetze. Es ist beispielsweise möglich neue, beliebige Malware oder Software auf den Zombie-Rechnern zu verteilen. Außerdem können Onlineumfragen oder Spiele manipuliert werden [17, 23].

2.8 Erkennungstechniken

Da sich viele Botnetze sehr unterscheiden, gibt es fast eben so viele Techniken diese zu erkennen. In dieser Arbeit werden daher nur die Grundbegriffe erklärt. Die Techniken lassen sich laut Tyagi und Aghila [23] in zwei Bereiche einteilen, honeypotbasierte Methoden und passive Überwachung des Netzwerkverkehrs.

2.8.1 Honeypotbasierte Methoden

Als einen Honeypot bezeichnet man im Bereich der Computersicherheit ein Programm, welches eine angreifbare Software oder ein verwundbares System simuliert. Der Angreifer wird durch die Schwachstellen darauf aufmerksam und versucht seinen Bot auf dem System zu installieren. Das ist genau das, was der Ersteller des Honeypots bezwecken will, denn alle ausgeführten Kommandos und Schritte werden aufgezeichnet. Auch evtl. geladene Binarys mit der Schadsoftware werden gespeichert. Somit ist es möglich neue Bots über ihre Verhaltensweisen aufzuspüren und diese nach dem Angriff genauer zu analysieren [23].

2.8.2 Passive Überwachung

Bei der passiven Netzwerkverkehrsüberwachung wird mithilfe verschiedener Klassifizierungstechniken nach Auffälligkeiten im Netzwerkverkehr gesucht. Diese Techniken klassifizieren auf Grund von Verhalten, DNS oder Data-Mining. Zur Überwachung wird in der Regel ein Intrusion Detection System (IDS), eine Software zum Erkennen von Angriffen, verwendet.

Verhaltensbasierte Erkennung

Die verhaltensbasierte Erkennung wird nochmals in zwei Bereiche eingeteilt, die signaturbasierte Erkennung und die anomaliebasierte Erkennung [23].

- **Signaturbasierte Erkennung:** Es wird auf Information über schon bekannte Bots zurückgegriffen. Das IDS überprüft Dateien und Prozesse auf Muster bekannter Bots. Wird ein Schema erkannt, schlägt das System Alarm. Das Problem hierbei ist jedoch, dass keine neuen Botnetze erkannt werden können, da keine Daten dafür vorhanden sind.
- **Anomaliebasierte Erkennung:** Hier wird nach Auffälligkeiten im Netzwerk gesucht. Dazu wird das normale Netzwerkverhalten als Vorlage gespeichert. Das IDS überprüft damit den Netzwerkverkehr und schlägt bei Ungereimtheiten Alarm.

Data-Mining basierte Erkennung

Beim Data-Mining können große Mengen von Verbindungsdaten analysiert, Unregelmäßigkeiten und Querverbindungen gefunden und somit Botnetze enttarnt werden. Die Technik hat sich dabei, genau wie die Botnetze, stark weiterentwickelt. So wird heute auf Techniken wie Clustering, Statistische Datenanalyse und neuronale Netze zurückgegriffen, um Anomalien zu bestimmen [23].

DNS-basierte Erkennung

Beim DNS-Erkennungsverfahren wird in den DNS-Informationen nach Anomalien gesucht. Dabei wird sich zunutze gemacht, dass sich der Bot (vgl. Abschnitt 2.4) immer wieder mit dem C&C Server verbindet. Um den richtigen Server zu finden, werden vom infizierten Rechner DNS-Anfragen durchgeführt. Da die C&C Serveradressen meistens von Dynamic-DNS (DDNS) Anbietern gehostet werden, entstehen so Unregelmäßigkeiten, die erkannt werden können. Diese Technik der DNS-basierten Erkennung kann jedoch einfach durch das Manipulieren der DNS-Abfragen verhindert werden, außerdem kommt es zu vielen Fehlalarmen. Eine Verbesserung wurde erreicht, als 2006 eine neue Technik vorgestellt wurde, die die NXDOMAIN-Antwortraten auswertet. Weil durch diese neue Methode DNS-Anfragen leichter auf

DDNS zurückzuführen sind, gibt es auch nicht so viele falsche Erkennungen wie bei der vorherigen Methode [10, 23].

Kapitel 3

Das Mirai-Botnetz

Inhalt des Kapitels

Ab hier wird sich auf ein spezielles Botnetz bezogen, das Mirai-Botnetz, welches sich hauptsächlich über ungeschützte IoT Geräte, die auf Linux basieren, wie z. B. IP-Überwachungskameras oder mit dem Internet verbundene Heizungssteuerungen, verbreitet. Anfangs wird die Geschichte des Mirai-Botnetzes geschildert. Danach wird der Aufbau von Mirai mit seinen verschiedenen Komponenten und deren Funktionen beschrieben. Anschließend wird die Architektur und die Kommunikation zwischen den Komponenten, sowie der Lebenszyklus des Mirai-Bots dargestellt. Danach werden die ausführbaren Angriffe erklärt. Um das Infektionsrisiko zu bestimmen, werden abschließend Daten, die mithilfe eines Honeypots gesammelt wurden, ausgewertet.
Zu beachten ist, dass hier die Grundform des Mirai-Netzes, also die Form, die Ende 2016 veröffentlicht wurde, behandelt wird. Mittlerweile gibt es auch weiterentwickelte Versionen und Abwandlungen mit mehr Angriffsvektoren oder anderen Verbesserungen auf die in dieser Arbeit nicht eingegangen wird.
Die meisten Erkenntnisse des Kapitels wurden dabei entweder aus dem Source Code von Mirai [25] oder aus dem dazugehörigen Forum-Post [26, 27] entnommen.

3.1 Geschichte

Die Geschichte ist chronologisch aufgeführt, analog dazu findet man unter Tabelle 3.1 eine Zeitleiste mit den wichtigsten Ereignissen. Anfang Juli 2016 wurde unter dem Pseudonym „Anna Senpei “ der/die vermeintliche Ersteller/in das erste Mal in einem englischsprachigen Hacker-Forum (hackforum.net) aktiv. Dieser Zeitpunkt wird von Apvrille [28] auch als der Anfang der Entwicklung des Mirai-Botnetzes vermutet. Mirai ist das japanische Wort für Zukunft, so lässt sich vermuten, dass „Anna Senpai “mit seinem/ihrem Botnetz etwas zukunftsweisendes erschaffen wollte. Nach ca. drei Monaten, am 20.09.2016 gab es dann den ersten DDoS-Angriff, der auf das Mirai-Botnetz zurückzuführen ist. Das Opfer der Attacke war der Blog des Cybersecurity Reporters Brian Krebs, krebsonsecurity.com. Krebs geht davon aus, dass der Angriff in Zusammenhang mit seinen zuvor gemachten Recherchen zu vDoS, einem DDoS-Dienst, der in Israel beheimatet ist, steht. Der Content Delivery Network (CDN) Anbieter von Krebs, Akamai, berichtete über die Attacke von einem rekordhohen Datendurchsatz von 665 Gigabit pro Sekunde. Die Attacke konnte aber erfolgreich abgewehrt werden [29]. Nachdem die Attacke aber noch weitere zwei Tage anhielt, entschloss sich das Unternehmen den Blog nicht weiter zu schützen und ihn vom Netz zu nehmen [30]. Erst nach knapp vier Tagen konnte die Seite wieder online gehen. Dies war nur unter dem Schutz von Googles „Project Shield “, einem Dienst von Google, der kostenlos Journalisten und Nachrichtenseiten vor DDoS Attacken schützt, möglich [31]. Kurz darauf startete ein mehrtägiger Angriff auf den französischen Telekommunikationsanbieter und Internetdienstleister OVH, welche die vorherige Attacke mit einem Spitzendurchsatz von 1,1 Terrabit pro Sekunde (Tbps) nochmal übertraf [32].
Wenig später, am 30. September 2016 veröffentlichte „Anna Senpei“ im oben genannten Hacker-Forum den gesamten Programmcode, auf dem das Mirai-Botnetz beruht, inklusive einer Anleitung wie dieses zu installieren ist [5]. Am 21. Oktober startete ein Mirai-Botnetz die bis heute größte DDoS-Attacke, die jemals aufgezeichnet wurde. Nach der Veröffentlichung des Codes ist nicht ganz klar, ob es sich hier noch um das ursprüngliche Netz oder ein neues handelt. Der Angriff richtete sich gegen Dyn, einen Internetdienstleister, der einen Großteil der DNS Infrastruktur kontrolliert. Dies führte dazu, dass Webseiten mehrerer Kunden von Dyn, unter anderem große Konzerne wie Paypal, Reddit, Amazon und Twitter zeitweise nicht

erreichbar waren [2, 3].
Laut Berichten von Dyn wurde der Angriff mit ca. 100 000 Teilnehmern durchgeführt und erreichte einen Datendurchsatz von bis zu 1,2 Tbps [4].

Tabelle 3.1: Zeitleiste wichtiger Ereignisse

Juli 2016	Anfang der Entwicklungsphase?
20. September 2016	DDoS-Attacke auf KrebsOnSecurity.com
22.-29. September 2016	DDoS-Attacke auf OVH
30. September 2016	Veröffentlichung des Programmcodes
21. Oktober 2016	DDoS-Attacke auf DYN

3.2 Aufbau und Ablauf

Hier wird auf die einzelnen Komponenten des Mirai Botnetzes eingegangen und deren Funktionsweise genauer erklärt. Es besteht aus einem C&C Server, der Bot-Software selbst, einem sogenannten Loader und aus weiteren kleinen Programmen, welche im Folgenden abschnittsweise erklärt werden. Die Ordnerstruktur sieht wie folgt aus:

```
/ (Hauptordner)
├── loader ... Enthält nötige Dateien des
│              loaders.
│   ├── bins ... Enthält binarys für den         .
│   │            Download-Helfer.
│   └── src
└── mirai
    ├── bot ... Enthält den Source-Code der
    │           Bots.
    ├── cnc ... Enthält den Source-Code des C&C
    │           Servers.
    └── tools ... Enthält den Source-Code für
                  weitere Tools.
```

3.2.1 Bot

Hier wird der Code des Bots genauer analysiert. Dieser ist die eigentliche Schadsoftware, die auf den Opfer-Rechnern ausgeführt wird. Um der Erkennung des Bots vorzubeugen, werden alle benutzten Adressen (z. B. zum C&C Server) und Befehle über einen XOR-Algorithmus (siehe Source Code 3.2.1) mit einem bestimmten Key verschlüsselt und in einer Tabelle fest einprogrammiert. Nur wenn auf einen Wert dieser Tabelle zugegriffen werden muss wird dieser mit dem gleichen Schlüssel decodiert und kann somit benutzt werden. Um die gewünschte Adresse zu verschlüsseln, kann das Programm *enc* aus dem Tool-Ordner genutzt werden. Der Standardschlüssel der Codierung lautet *0xdeadbeef*.

Source Code 3.2.1: Ver- und Entschlüsselungs-Algorithmus der fest einprogrammierten Daten

```
uint32_t table_key = 0xdeadbeef;
...
uint8_t k1 = table_key & 0xff,
k2 = (table_key >> 8) & 0xff,
k3 = (table_key >> 16) & 0xff,
k4 = (table_key >> 24) & 0xff;

for (i = 0; i < val->val_len; i++)
{
   val->val[i] ^= k1;
   val->val[i] ^= k2;
   val->val[i] ^= k3;
   val->val[i] ^= k4;
}
```

Nach dem Start des Bots überprüft das Programm über die Konstante `DEBUG` ob es im Debug Modus ausgeführt wird. Ist dies der Fall werden verschiedene Ausgaben zur Fehlerkorrektur während des Programmablaufs gemacht, wie z. B. dass die Verbindung zum C&C Server erfolgreich hergestellt worden ist. Wenn die Konstante nicht definiert wurde, wird kurz nach dem starten die Binary Datei des Bots gelöscht.

Die Datei ist also nicht mehr auf dem Zombie-Rechner zu finden, während der Bot weiterhin ausgeführt wird. Außerdem wird der *watchdog*, also der Prozess, der das System auf Fehler überwacht, daran gehindert den Computer neu zu starten. Über das Abfangen des `SIGTRAP` Signals wird zusätzlich verhindert, dass Debugger wie der GNU-Debugger(gdb) die echte C&C Adresse herausfinden können.

Source Code 3.2.2: Start des Mirai Bots

```
// Delete self
unlink(args[0]);

// Signal based control flow
sigemptyset(&sigs);
sigaddset(&sigs, SIGINT);
sigprocmask(SIG_BLOCK, &sigs, NULL);
signal(SIGCHLD, SIG_IGN);
signal(SIGTRAP, &anti_gdb_entry);

// Prevent watchdog from rebooting device
if ((wfd = open("/dev/watchdog", 2)) != -1 ||
(wfd = open("/dev/misc/watchdog", 2)) != -1)
{
   int one = 1;
   ioctl(wfd, 0x80045704, &one);
   close(wfd);
   wfd = 0;
}
chdir("/");
```

Nun wird über den Port *48101*, den Standardport für das Melden neuer Opfer-Rechner, nach schon laufenden Instanzen von Mirai gesucht. Werden Prozesse gefunden werden diese ausgeschaltet, um selbst den Port übernehmen zu können. Um den Prozess zu verschleiern, wird dieser durch das Ändern des Prozessnamens in einen zufällig generierten Namen geändert. Danach werden durch den Aufruf

der `attack_init()` Funktion die Angriffsvektoren initialisiert. Mehr zu den verschiedenen hier genutzten Angriffsmethoden können unter Abschnitt 3.5 gefunden werden. Nachfolgend wird `kill_init()`, eine Funktion zum Abschalten störender Prozesse aufgerufen. Hier teilt sich der Prozess in zwei Prozesse auf. Der Hauptprozess kehrt zum eigentlichen Programm zurück, der sogenannte Kindprozess bleibt in der kill-Funktion. In dieser schaltet er Prozesse ab, die Port 23 belegen, um selbst über diesen zu verfügen. Anschließend wird in einer Schleife nach schon vorhandener Malware auf dem Gerät gesucht. Dabei wird auf sogenanntes „Memory Scraping“ zurückgegriffen, bei dem der Arbeitsspeicherbereich eines jeden Prozesses nach bestimmten Schlüsselwörtern durchsucht wird, welche die verschiedenen Malwaretypen identifizieren sollen [33]. Werden solche Muster im Prozess gefunden wird dieser zerstört. So sollte es Mirai möglich sein, das Potential der Attacke zu maximieren, da es nicht mit anderer Malware um Rechenleistung des PCs konkurrieren muss. Abschließend verhindert der Kindprozess, dass der Arbeitsspeicher erneut gescannt werden kann.

Scanner

Der Hauptprozess startet währenddessen über den Aufruf von `scanner_init()` einen weiteren Kindprozess während er selbst in die Hauptfunktion zurückkehrt. Nach dem Einrichten des TCP- und IPv4-Headers wird die Passwortliste vom Kindprozess initialisiert. Die Passwörter sind wie in Source Code 3.2.3 in verschlüsselter Form im Programmcode hinterlegt. Bei den 62 Passwörtern handelt es sich um Standardlogins, die Hersteller für ihre Geräte vergeben und oftmals nicht ändern. Die komplette Liste kann in Anhang A eingesehen werden. Somit nutzt Mirai direkt die Fahrlässigkeit der Hersteller aus.

Source Code 3.2.3: Initialisierung der Passwortliste

```
    // Set up passwords
    add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x41\x11\x17\x13\x13",
↪   10);                    // root     xc3511
    add_auth_entry("\x50\x4D\x4D\x56", "\x54\x4B\x58\x5A\x54", 9);
↪   // root     vizxv
```

```
    add_auth_entry("\x50\x4D\x4D\x56", "\x43\x46\x4F\x4B\x4C", 8);
↪   // root      admin
    add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C", 7);
↪   // admin     admin
```

Um die Passwörter zu entschlüsseln und zu benutzen, wird, wie auch bei der anfangs erwähnten Tabelle eine Verschlüsselung über byteweise XOR-Verknüpfung durchgeführt (siehe Source Code 3.2.4). Dazu wird auch hier der Schlüssel *0xdeadbeef.* verwendet.

Source Code 3.2.4: Entschlüsslungsalgorithmus der Anmeldedaten

```
for (i = 0; i < *len; i++)
{
    cpy[i] ^= 0xDE;
    cpy[i] ^= 0xAD;
    cpy[i] ^= 0xBE;
    cpy[i] ^= 0xEF;
}
```

Nach der Initialisierung kommt der Kindprozess in eine Endlosschleife oder auch *Main-Loop*, in der er nach neuen Opfern für das Botnetz sucht. Dazu wird eine zufällige IP-Adresse generiert, wobei ein paar IP-Adressen, beispielsweise die *Loopback*-Adresse 127.0.0.1 oder die des *US Postal Service*, ein Postdienstleister der USA, von der Generierung ausgeschlossen sind. Da unter anderem auch IP Adressen des Verteidigungsministerium der Vereinigten Staaten ignoriert werden, ist zu vermuten, dass der Autor verhindern wollte, dass das Botnetz zu viel Aufmerksamkeit auf sich zieht. Die gesamte Liste der IP Adressen, die nicht generiert werden, ist in Anhang B zu finden.
Nachdem die IP ausgewählt wurde, wird überprüft ob diese Adresse erreichbar ist. Ist dies nicht der Fall wird eine andere IP-Adresse generiert. Konnte erfolgreich eine Verbindung hergestellt werden, wird nun ein Telnet-Client simuliert, der versucht sich auf Port 23, also dem Standard-Port für Telnet, auf dem Zielrechner einzuloggen. Dazu

wird ein Wörterbuchangriff mit den vorher definierten Standardlogindaten ausgeführt. Sind die Logindaten falsch wird eine andere Benutzer/Passwort-Kombination ausprobiert. Sind die Daten korrekt wird über das Ausführen von einigen Befehlen sichergestellt, dass die Authentifizierung wirklich richtig war. Ein Verbindungsablauf kann unter Abbildung 3.2.1 eingesehen werden. Hat alles funktioniert wird nun die IP-Adresse, der Port, sowie die funktionierenden Logindaten an den Loader weitergeleitet, welcher sich dann beim Computer des Opfers einloggt und versucht die Bot-Software darauf zu installieren (siehe Abschnitt 3.2.3). Dann startet der Kindprozess am Anfang seiner Main-Loop neu um nach mehr potentiellen Zombie-Rechnern zu suchen.

```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

root@server:~# enable
enable .
enable :
enable [
enable alias
enable unalias
enable unset
enable wait
root@server:~# system
bash: system: command not found
root@server:~# shell
bash: shell: command not found
root@server:~# sh
root@server:~# /bin/busybox MIRAI
MIRAI: applet not found
```

Abbildung 3.2.1: Verbindungsaufbau zwischen Bot und Opfer-Rechner

Der Hauptprozess erreicht währenddessen auch seine Main-Loop. In dieser verbindet er sich mit dem C&C Server und meldet sich somit als zur Verfügung stehender Zombie-Rechner an. Jetzt wird auf Befehle des C&C Servers gewartet. Sobald einer empfangen wurde wird dieser geparst und wieder von einem Kindprozess ausgeführt. Damit besonders viele Rechnerarchitekturen wie z. B. ARM oder MIPS unterstützt werden, wird zum kompilieren der Schadsoftware auf sogenannte Cross-Compiler gesetzt. Diese erlauben es, Programme auf einem bestimmten System für ein anderes System bzw. eine andere Architektur zu kompilieren. Somit ist es möglich mit

nur einem Rechner die Schadsoftware für alle gewünschten Systeme zu erstellen. Die Binary-Dateien werden in diesem Fall nach dem Schema *mirai.architektur* also beispielsweise *mirai.arm* oder *mirai.mips* gespeichert. Dies ermöglicht es mithilfe des Architekturnamens gezielt die richtige Schadsoftware auf den zu infizierenden Computer herunterzuladen.

3.2.2 Command and Control Struktur

Die C&C Struktur ist zum Überwachen der schon infizierten Rechner sowie zum Steuern von diesen gedacht. Der gesamte Code des Server ist in GO, einer recht neuen Programmiersprache, geschrieben.

MySQL-Datenbank

Um den Kontroll-Server zu benutzen, muss zunächst eine MySQL Datenbank eingerichtet werden, welche die Tabellen *users*, *history* und *whitelist* enthält. Die Struktur ist in Abbildung 3.2.2 zu sehen.
In der Tabelle *users* lassen sich die Benutzer für den C&C Server einrichten. Durch Spaltennamen wie *last_paid* oder *max_bots* fällt auf, dass das Botnetz nicht nur auf einen Botmaster zugeschnitten ist. Stattdessen bekommt dieser die Möglichkeit, neue User zu erstellen und somit Bots aus dem Botnetz zu vermieten (*last_paid*). Über die Spalte *max_bots* lässt sich einstellen wie viele Bots dem Mieter zu Verfügung gestellt werden. Das Feld *duration_limit* gibt die maximal erlaubte Dauer beim Ausführen eines Angriffs an. Außerdem lässt sich über *cooldown* die Zeit, die zwischen 2 Angriffen liegen muss, einstellen. So kann sich eine Preisstruktur entwickeln: Je mehr Bots und je längere Angriffsdauer desto teurer wird das Mieten der Bots. Durch eine längere Cooldown-Phase kann der Preis wieder verringert werden. Durch die *whitelist* Tabelle lassen sich Netzmasken definieren, welche nicht angegriffen werden können. In der Tabelle *history* werden die getätigen Befehle mit der zugehören Benutzer-ID gespeichert. In Anhang C kann der SQL-Code zum Erstellen der Tabellen gefunden werden.

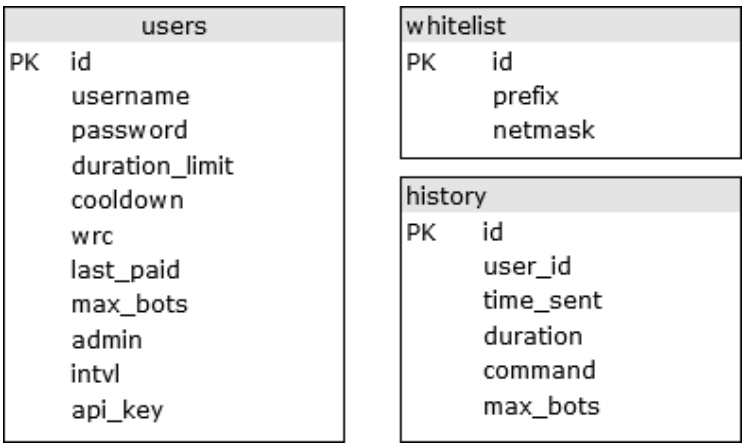

Abbildung 3.2.2: Tabellenstruktur der Mirai Datenbank

C&C Server

Nach dem Start des C&C Servers wird als erstes eine Verbindung mit der oben beschriebenen MySQL-Datenbank hergestellt. Nun wird auf eine eingehende Verbindung auf Port 23 oder Port 101 gewartet.
Bei der Verbindung über den Port 23 wird unterschieden ob es sich um einen Bot oder einen Benutzer handelt. Wie in Source Code 3.2.5 zu sehen geschieht diese Unterscheidung über die ersten 4 empfangenen Bytes. Sind diese alle Null geht das Programm davon aus, dass es sich um einen Bot handelt und fügt diesen zu den aktiven Bots des Netzwerks hinzu. Ist dies nicht der Fall wird davon ausgegangen, dass ein Admin oder Benutzer versucht sich einzuloggen.

Source Code 3.2.5: Unterscheidung zwischen Bot und Benutzer im C&C Server von Mirai.

```
    if l == 4 && buf[0] == 0x00 && buf[1] == 0x00 && buf[2] == 0x00
↪ {
        if buf[3] > 0 {
            string_len := make([]byte, 1)
            l, err := conn.Read(string_len)
            if err != nil || l <= 0 {
                return
            }
            var source string
```

```
            if string_len[0] > 0 {
               source_buf := make([]byte, string_len[0])
               l, err := conn.Read(source_buf)
               if err != nil || l <= 0 {
                  return
               }
               source = string(source_buf)
            }
            NewBot(conn, buf[3], source).Handle()
         } else {
            NewBot(conn, buf[3], "").Handle()
         }
      } else {
         NewAdmin(conn).Handle()
      }
```

Dabei wird der User, der sich eingeloggt hat, von der russischen Nachricht *я люблю куриные наггецы* begrüßt, welche auf englisch so viel wie „I love chicken nuggets“ heißt. Danach wird man ebenfalls auf russisch aufgefordert, den Benutzernamen und das Passwort einzugeben. Der Ablauf ist in Abbildung 3.2.3 zu sehen.

```
я люблю куриные наггетсы
пользователь: Botmaster
пароль: *********

проверив счета... |
[+] DDOS | Succesfully hijacked connection
[+] DDOS | Masking connection from utmp+wtmp...
[+] DDOS | Hiding from netstat...
[+] DDOS | Removing all traces of LD_PRELOAD...
[+] DDOS | Wiping env libc.poison.so.1
[+] DDOS | Wiping env libc.poison.so.2
[+] DDOS | Wiping env libc.poison.so.3
[+] DDOS | Wiping env libc.poison.so.4
[+] DDOS | Setting up virtual terminal...
[!] Sharing access IS prohibited!
[!] Do NOT share your credentials!
Ready
Botmaster@botnet#
```

Abbildung 3.2.3: Logindialog vom C&C Server des Mirai Botnetzes.

Nun hat der Benutzer die Möglichkeit Befehle auszuführen. Einem User mit Adminrechten, also in der Regel der Botmaster, werden dabei mehr Befehle angeboten als den normalen Nutzern. So kann beispielsweise über den Befehl *adduser* ein neuer Benutzer in der Datenbank angelegt oder über *botcount* die Anzahl der Bots ausgegeben werden. Zudem können die Bots in Gruppen eingeteilt werden. Den normalen Benutzern bleibt hier hauptsächlich die Option einen Angriff mit den gemieteten Bots auszuführen. So kann man z. B. durch den Befehl *udp 8.8.8.8 60* eine 60 sekündige UDP-Float-Attacke auf die angegebene IP-Adresse ausführen. Wichtig dabei ist, dass die C&C Struktur keine Möglichkeit bietet die Angriffe abzubrechen. Mehr zu den verschiedenen Angriffsmethoden kann unter Abschnitt 3.5 gefunden werden.

Wie schon anfangs erwähnt wartet der Server nicht nur auf Verbindungen auf Port 23 sondern auch auch Port 101. Über diesen Port ist es möglich die API (Application Programming Interface, deutsch: Programmierschnittstelle) des Servers anzusprechen. Das ermöglicht es dem Benutzer sich mit der Angabe seines API-Keys zu authentifizieren und somit direkt Befehle an den C&C Server zu schicken. Der oben genannte Befehl kann beispielsweise direkt über das Senden von `1234|udp 8.8.8.8 60` durchgeführt werden, wobei 1234 hier dem API-Key entspricht. Wurde das Kommando erfolgreich angenommen bekommt man ein „OK“ als Antwort. So ist es denkbar benutzerfreundliche Webinterfaces zu erstellen, über die der Kunde das Botnetz auf Knopfdruck steuern kann.

3.2.3 Scanlisten und Loader

Die Funktion neue Computer zu infizieren wird durch *scanlisten* aus dem Tool-Ordner sowie dem *loader* ermöglicht.

Scanlisten

Das Programm *scanlisten* ist für das Empfangen der Scannergebnisse von den Zombie-Rechnern, die nach neuen infizierbaren Rechnern suchen, zuständig. Dazu wird auf Port 48101 auf eingehende Verbindungen gewartet. Werden neue erkannt, werden die empfangenen Informationen Byte für Byte eingelesen, und die Werte voneinander getrennt. So ist es möglich die IP-Adresse, den Port und die

Anmeldedaten aus der Nachricht des Bots zu extrahieren. Diese werden dann im Format `IP:Port Username:Passwort` auf die Standardausgbe (stdout) geschrieben, also in der Regel auf dem ausführenden Terminal ausgegeben.

Loader

Der *loader* ist für das Infizieren neuer Rechner zuständig indem er sich bei diesen einloggt, die Schadsoftware herunterlädt und ausführt. Dazu wird wie folgt vorgegangen:
Anfangs wird überprüft, ob benötigte Binary-Dateien für den Downloadhelfer vorhanden sind. Damit dieser auch für die verschiedenen Rechnerarchitekturen zur Verfügung steht, muss er zuvor, wie schon bei den Bots, über einen Cross-Compiler kompiliert werden. Der Downloadhelfer kommt später zum Einsatz, wenn die Befehle wget und tftp nicht zur Verfügung stehen.
Wurden die Dateien erfolgreich gefunden, werden Eingaben aus der Standardeingabe (stdin) gelesen. Diese werde im Format `IP:Port Username:Passwort` erwartet, also genau so wie *scanlisten* seine Daten in die Standardausgabe schreibt. Sind Daten vorhanden, werden diese geparst und somit auf das richtige Format überprüft. Ist dies erfolgreich versucht der *loader* sich am Zielrechner anzumelden und diesen mit dem Bot zu infizieren. Dazu werden separate *worker*-Prozesse erstellt, welche über ein Epoll, ein System für Ereignisbenachrichtigungen, auf sogenannte Events warten und diese ausführen. Diese Arbeiterprozesse stellen dann eine Verbindung mit dem Opfer her und versuchen sich mit den übermittelten Anmeldedaten einzuloggen. Gelingt dies, werden nun verschiedene Befehle über die *BusyBox*, ein Programm, dass auf vielen Linux Geräten vorhanden ist und grundlegende UNIX-Dienstprogramme wie z. B. den echo-Befehl bereitstellt, ausgeführt. Dabei wird auch die Architektur des Zielrechners bestimmt. Nun muss der Bot nur noch hochgeladen und ausgeführt werden.
Dazu wird überprüft welche Methode zum Upload zur Verfügung steht. Ist der Befehl *wget* oder *tftp* über die BusyBox verfügbar wird einer von diesen genutzt um die Schadsoftware auf den Opfer-Rechner zu laden. Ist dies nicht der Fall wird der echo-Befehl ausgenutzt, um den zur Architektur passenden Downloadhelfer auf dem

entfernten Computer zu erstellen, um darüber den Bot hochzuladen. Abschließend wird der Mirai-Bot gestartet und das System aufgeräumt, indem der Downloadhelfer und andere erstellte Dateien gelöscht werden. Dieser Vorgang wiederholt sich für alle Daten, die vom *loader* in der Standardeingabe eingelesen werden können, danach wird dieser beendet.

Da es sehr umständlich ist die *scanlisten*-Ausgaben zu kopieren und beim *loader* einzufügen besteht in Linux die Möglichkeit, die empfangenen Daten in eine Datei umzuleiten (`./scanlisten > daten.txt`) und diese dann über eine Pipe im *loader* zu laden (`cat daten.txt | ./loader`). Somit lassen sich die Abläufe noch mehr automatisieren.

3.3 Architektur und Kommunikation

In diesem Abschnitt soll geklärt werden welche Hardware-Komponenten für das Mirai-Botnetz benutzt wurden und welche Architektur damit aufgebaut wird. Zur Veranschaulichung dient dazu Abbildung 3.3.1. Dabei wird versucht den im Forum-Post von „Anna Senpei“ beschriebenen Aufbau, den der Ersteller nach eigenen Angaben selbst nutzte, nachzustellen. Der Urheber benötigte dafür laut dem Forum-Post zwei Virtual Private Server (VPS) sowie vier normale Server. Diese sind wie folgt aufgeteilt:

- Ein VPS für die MySQL Datenbank der Kontrollstruktur. Hier sollte laut Senpai darauf geachtet werden, dass der Anbieter „extrem kugelsicher“ ist.
- Ein VPS für das Sammeln von Scannergebnissen der Bots, sowie zum Weiterleiten von diesen an die Loader
- Drei Server für die Loader, d. h. für das Infizieren von potentiellen Bots mithilfe der Scannergebnisse.
- Ein Server für die C&C Anwendung.

Wie in Abbildung 3.3.1 zu sehen ist verbindet sich der Bot über Port 23 direkt mit dem C&C Server, was zeigt, dass es sich hier um eine zentralisierte Architektur handelt, nur, dass noch mehr Komponenten vorhanden sind. Da im Bot nur die

Möglichkeit besteht eine einzige Domain für den Kontroll-Server zu definieren, und diese auch nicht einfach geändert werden kann, ist davon auszugehen, dass hier ein „bulletproof“ DNS-Anbieter genutzt wurde. Ansonsten würde man bei der Abschaltung der Domain die Kontrolle über das Botnetz verlieren.
Für die Verbindung zwischen Bots und dem VPS, der die Scannergebnisse empfängt, wird auch nur eine Domain hinterlegt. Deshalb kann man auch bei dieser Verbindung davon ausgehen, dass auf einen DNS-Anbieter zurückgegriffen wurde, der sämtliche Missbrauchsmeldungen ignoriert. Das Verteilungsprogramm, das auf dem gleichen Server läuft, verteilt laut dem Ersteller die gefundenen Ergebnisse gleichmäßig an die drei Load-Server. Diese bauen eine Telnet-Verbindung mit den Opfern auf, laden die Schadsoftware auf den Computer herunter und führen diese letztendlich aus.
Bei der Kommunikation benutzt das Mirai Botnetz keine der unter Abschnitt 2.5.1.1 aufgeführten üblichen Methoden. Stattdessen wird zum Übertragen von Befehlen vom C&C Server zum Bot auf eine binäre Übertragung gesetzt. Das gleiche Verfahren wird auch beim Melden der Scannergebnisse genutzt. Beim Scannen selbst wird zum Suchen verfügbarer Ports auf das Transmission Control Protocol (TCP) gesetzt, um sich anzumelden wird dann das Telnet-Protokoll verwendet.

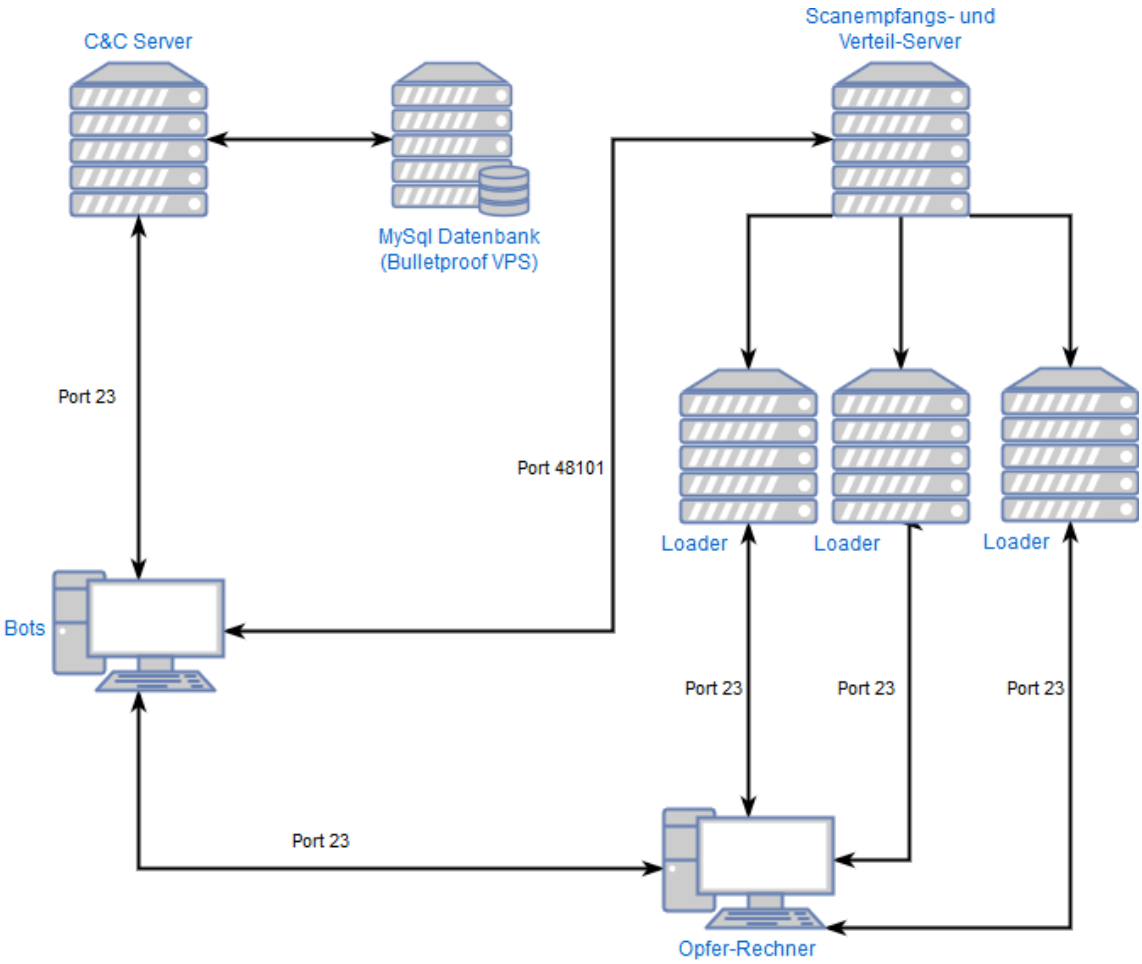

Abbildung 3.3.1: Topologie des Mirai-Botnetzes: Hier wird für die oben beschriebenen Module jeweils ein eigener Server oder zumindest ein VPS genutzt. Der Opfer-Rechner sowie der Bot-Rechner stehen hier jeweils für mehrere Systeme.

3.4 Lebenszyklus des Mirai-Bots

Der Lebenszyklus, der in Abschnitt 2.4 beschrieben ist trifft hier nicht vollständig zu. Er muss vor allem um die parallelen Abläufe des Mirai Bots wie z. B. die Scan-Funktion ergänzt werden.
Der daraus resultierende Lebenzyklus wird in Abbildung 3.4.1 dargestellt und die Phasen im Folgenden erklärt:

- **Infektionsphase:** Ein Bot versucht sich in dieser Phase mit einem der gespeicherten Passwörter beim Rechner des Opfer anzumelden. Gelingt dies werden die Daten an den Loader weitergegeben

- **Injektionsphase:** In der Injektionsphase loggt sich der Loader im potentiellen Zombie-Rechner ein, lädt die Binär-Dateien des Bots herunter und führt diese aus.

- **Initialisierungsphase:** In dieser Phase führt der eben gestartete Bot anfängliche Arbeiten durch wie z. B. das Initialisieren der Angriffsvektoren. Außerdem werden weitere Prozesse für den parallelen Programmablauf gestartet.

- **Verbindungsphase:** Wie auch oben beschrieben verbindet sich der Bot in dieser Phase mit dem C&C Server, und wartet dann auf Befehle. Sollten keine Befehle empfangen werden, wird sich nach einem Timeout erneut mit dem C&C Server verbunden.

- **Angriffsphase:** In der Angriffsphase werden empfangene DDoS Befehle ausgeführt. Da hierfür ein extra Prozess erstellt wird, kann der Hauptprozess schon wieder in die Verbindungsphase zurückkehren und auf weitere Befehle warten.

- **Scanphase:** Hier wird nach neuen Opfern für das Botnetz gesucht. Werden welche gefunden wird versucht sich mithilfe der hinterlegten Standardpasswörter (siehe Anhang A) einzuloggen.

- **Reportphase:** Ist der Zugriff auf das System erfolgreich hergestellt, werden in der Reportphase die Scandaten an den Scanempfangsserver gesendet. Danach wird nach weiteren Opfern gesucht.

- **Prozessscanphase:** Hier werden, soweit vom Betriebssystem erlaubt, die laufenden Prozesse auf bestimmte Schlüsselwörter durchsucht. Somit kann beispielsweise feindliche Schadsoftware wie der „Q-Bot“ aufgespürt werden.

- **Prozesskillphase:** Hier werden gefundene, unerwünschte Prozesse beendet. Anschließend werden die Prozesse erneut auf Auffälligkeiten gescannt.

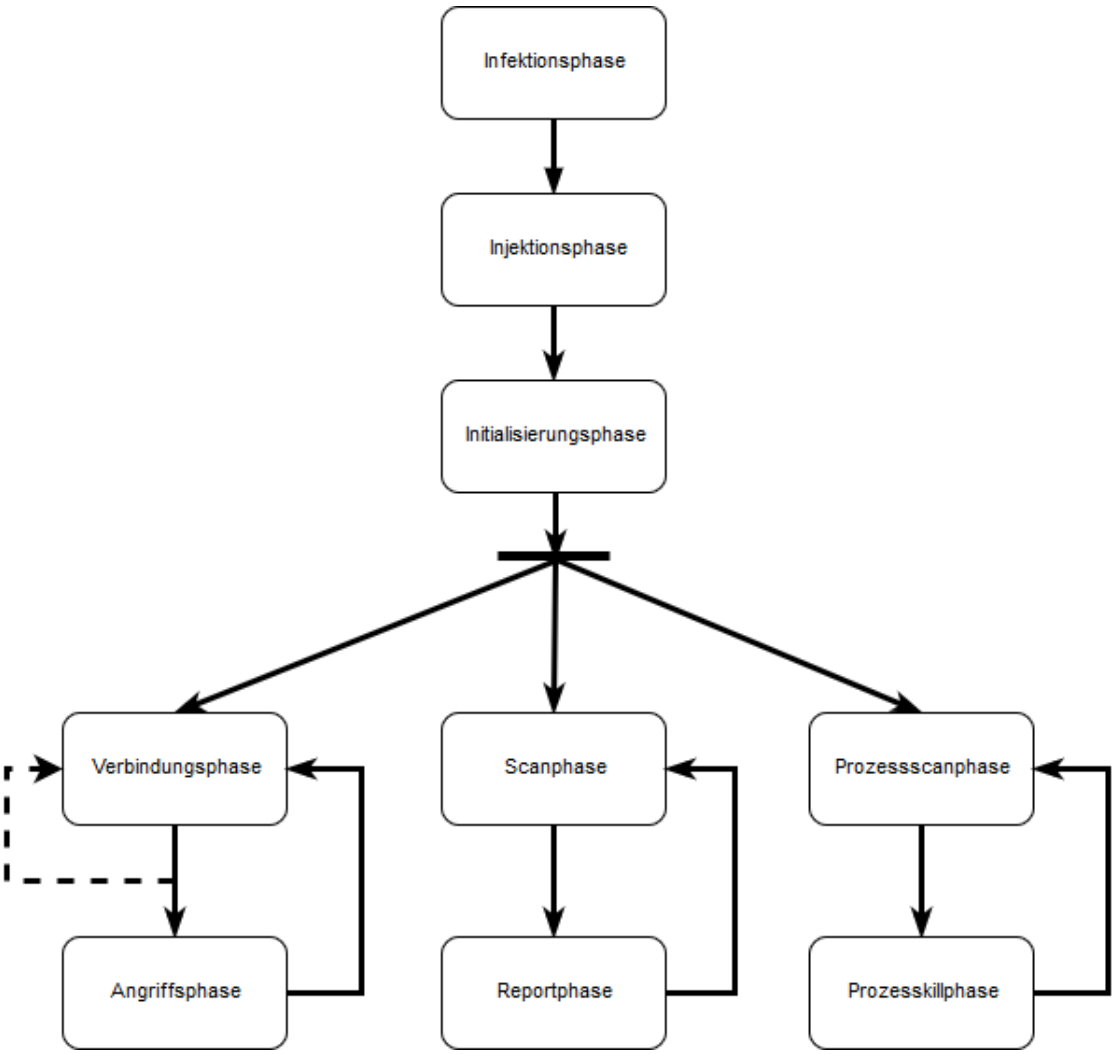

Abbildung 3.4.1: Lebenszyklus des Mirai Bots.

3.5 Angriffe

Das Mirai-Botnetz verfügt über eine Reihe verschiedener DDOS-Angriffe. Diese werden in diesem Kapitel erklärt. Abbildung 3.5.1 zeigt dazu die Liste der verfügbaren Angriffe, die das Terminal des C&C Servers liefert.

```
Botmaster@botnet# ?
Available attack list
vse: Valve source engine specific flood
syn: SYN flood
http: HTTP flood
stomp: TCP stomp flood
greip: GRE IP flood
greeth: GRE Ethernet flood
udpplain: UDP flood with less options. optimized for higher PPS
udp: UDP flood
dns: DNS resolver flood using the targets domain, input IP is ignored
ack: ACK flood
```

Abbildung 3.5.1: Verfügbare Angriffsmethoden des Mirai-Bots.

3.5.1 SYN-Flood

Bei dem SYN-Flood Verfahren wird eine TCP Verbindung mit dem Server aufgebaut. Dabei führen die beiden Teilnehmer normalerweise einen *Threeway-Handshake* durch um die Verbindung einzurichten. Dieses Verfahren wird in Abbildung 3.5.2 dargestellt und läuft folgendermaßen ab:

1. Der Client sendet ein Synchronize Paket (SYN) an den Server.
2. Der Server antwortet mit einem Synchronize-Acknowledge Paket (SYN-ACK) um den Empfang des SYN-Pakets zu bestätigen.
3. Der Client antwortet wiederum mit einem Acknowledge Paket (ACK), die Verbindung ist erfolgreich hergestellt.

Während der halboffenen Verbindung, d. h. während der Server auf den ACK des Clients wartet, speichert der Server die Adressen, den Port sowie den Status der Verbindung im Netzwerkstack. Dies wird sich bei der SYN-Flood Attacke zunutze gemacht indem der bösartige Client kein ACK an den Server übermittelt und die Verbindung so halboffen bleibt.
Noch während der Server auf die Antwort des Clients, hier des Mirai Bots, wartet fangen die Bots an SYN-Anfragen zu „fluten“ (flood), d. h. andauernd Synchronize-Pakete an den Server zu senden. Dadurch werden die Ressourcen, die der Server für den Netzwerkstack zum Speichern halboffener Verbindung bereitstellt, aufgebraucht. Ist dies der Fall, ist es nicht mehr möglich neue Verbindungen mit dem Server aufzubauen [34]. Der Angriff ist in Abbildung 3.5.3 dargestellt.

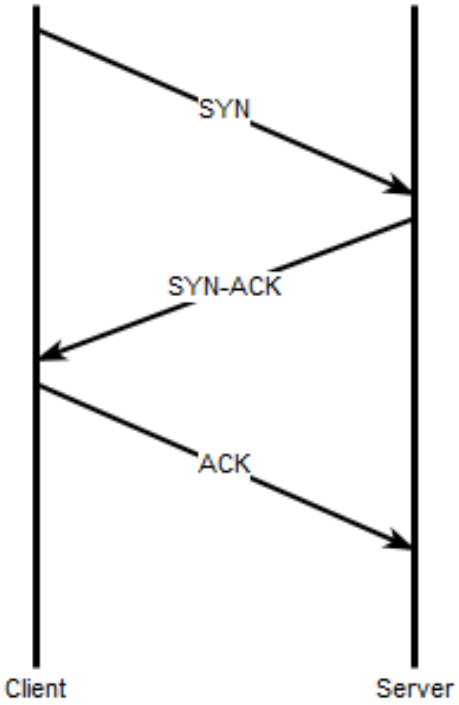

Abbildung 3.5.2: Normaler TCP-Handshake

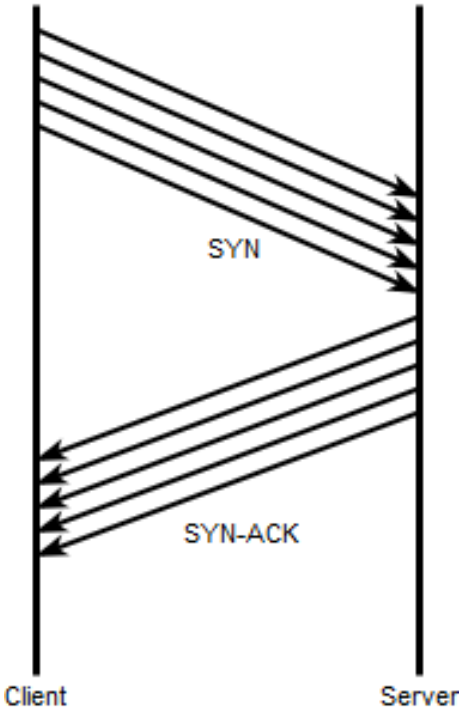

Abbildung 3.5.3: SYN-Flood Attacke

3.5.2 ACK-Flood

Auch beim ACK-Flood wird wie beim SYN-Flood der *Threeway-Handshake* (siehe auch Abbildung 3.5.2) von TCP Verbindungen ausgenutzt. Hier werden von den Bots aber statt den SYN-Paketen eine sehr hohe Anzahl an ACK-Paketen an den Server geschickt. Diese Pakete enthalten, wie auch beim Mirai-Netz standardmäßig eingestellt, zufällig generierte Informationen. Der Server versucht beim Empfang jedes ACK einer Verbindung zuzuordnen, was Rechenleistung benötigt. Ist die Anzahl der empfangenen Pakete zu hoch kann der Server überlastet werden,so dass keine normale Verbindung mehr möglich ist [35].

3.5.3 UDP-Flood

Beim UDP-Flood wird auf das User Datagram Protocol (UDP) zurückgegriffen, das im Gegensatz zu TCP nicht überprüft, ob die gesendeten Pakete angekommen sind (Fire-and-Forget). Hier senden die Bots UDP Pakete an zufällige Ports eines festgelegten Servers. Der Server überprüft beim Empfang, ob der angegebenen Port

von einer Anwendung benutzt wird. Ist dies nicht der Fall sendet er ein *Destination Unreachable* Paket zurück an den Client. Je mehr Anfragen der Client jetzt vom Botnetz bekommt, desto mehr solcher Pakete muss der Server verschicken. Das kann dazu führen, dass das System des Servers von der Menge an Anfragen überwältigt wird und somit anderen Clients nicht mehr antworten kann. Um sicher zu gehen, dass die Bots die Antwort des Servers nicht empfangen und um die Herkunft der Attacke zu verschleiern, kann außerdem die IP-Adresse im Header des Pakets manipuliert werden. Dies ist beim Mirai-Botnetz standardmäßig eingestellt. Die zur Verfügung stehende UDP-Plain-Flood Attacke funktioniert genau so. Es stehen lediglich weniger Optionen zur Verfügung da sie schon für eine hohe Paket-pro-Sekunde(PPS) Rate eingestellt wurden [36].

3.5.4 HTTP-Flood

Über die HTTP-Flood Attacke wird ein Webserver oder eine Web-Applikation angegriffen. Dazu senden die Bots *GET*- und *POST*-Anfragen an den Server. GET wird dabei hauptsächlich zum Holen von statischen Daten, wie z. B. Bildern benutzt. POST dagegen wird für das Anfragen von dynamischen Daten verwendet. Da die Bearbeitung jeder Anfrage System-Ressourcen des Servers beansprucht, wird mit dem „Fluten“ von GET oder POST durch die Bots versucht diese voll auszureizen. Dies kann am Besten geschafft werden, wenn jede einzelne Anfrage schon die dafür maximal zur Verfügung stehende Rechenleistung ausnutzt.Die komplexeren POST-Anfragen eignen sich dazu am besten, müssen aber auch spezieller eingestellt werden. Die GET-Anfragen sind dagegen einfacher zu erstellen und können innerhalb eines Botnetzes besser skalieren [37]. Das Mirai-Botnetz bietet für die HTTP-Flood Attacke beide Möglichkeiten an, benutzt standardmäßig aber GET-Anfragen.

3.5.5 DNS-Flood

Bei einer DNS-Flood Attacke wird nicht wie bei den bisherigen Attacken der Server des Opfers angegriffen, sondern der vom Opfer benutzte DNS-Server. Dieser löst den Domainnamen in die IP-Adresse für den Server bzw. Dienst des Opfers auf.
Die Bots senden dazu eine enorme Anzahl an DNS-Anfragen an den DNS-Server. Durch die große Masse an Anfragen wird versucht die Rechenkapazität von diesem aufzubrauchen. Daraus resultieren stark erhöhte Wartezeiten bei normalen DNS-

Anfragen bis hin zur Unerreichbarkeit des Dienstes, zumindest über die Domain. Das Fälschen des Paket-Headers mit zufälligen Daten kann außerdem dabei helfen, zu verhindern, dass der DDoS-Schutz die Attacke abwehren kann [38]. Diese Technik wird beim Mirai-Botnetz standardmäßig eingesetzt.

3.5.6 Valve-Source-Engine(VSE)Flood

Die VSE-Flood Attacke ist auf Spiele-Server ausgerichtet. Durch das „Fluten“ mit sogenannten *TSource Engine*-Anfragen, welche z. B. von *Counter Strike* Servern unterstützt werden, werden so viele Anfragen gestellt, dass nicht mehr alle beantwortet werden können. Hier wird sich nun die Verstärkung des Angriffs durch den Server selbst zunutze gemacht. Diese kommt dadurch zustande, dass die Antworten des Server größer sind als die eingegangenen Nachrichten. Durch diese Arbeitsweise verstärkt der Server, durch das Beantworten der vielen Anfragen, selbst die Effektivität der DDoS-Attacke. [39, 40].

3.5.7 GRE-Flood

GRE steht für Generic Routing Encapsulation, ein Netzwerkprotokoll, welches es erlaubt andere Protokolle einzukapseln und so in der Form eines Tunnels über das Internet Protokoll (IP) zu transportieren [41]. Das GRE-Protokoll kommt dabei häufig zum Tunneln von VPN-Verbindungen zum Einsatz und wird von Routern in der Regel einfach ungesehen weitergeleitet. Beim GRE-Flood Verfahren, einer sehr untypischen DDoS-Attacke, macht man sich das zunutze. Dies bietet den Bots die Möglichkeit ein GRE-Paket mit zufälligen gekapselten UDP-Daten, sogenannten Junk(zu dt. Müll) Daten, direkt zum Opfer der DDoS Attacke zu leiten. Der Computer des Opfers entpackt die Daten wieder und versucht diese zu interpretieren. Da dies Ressourcen des Rechners benötigt, kann es durch das „Fluten“ der GRE-Pakete zu einer Überlastung des Systems kommen [42].

3.5.8 STOMP-Flood

STOMP steht für „Simple Text Oriented Message Protocol“ und ist ein auf Applikations-Schicht laufendes, textbasiertes Protokoll. Es bietet Clients die Möglichkeit mit sogenannten „Message Brokern“, Programmen, welche die Kommuni-

kation in andere Protokolle übersetzen, zu kommunizieren. STOMP erlaubt es also, Programme, die auf unterschiedlichen Programmiersprachen basieren, Nachrichten auszutauschen. Dazu kommt als Grundlage der Verbindung das Transmission Control Protocol (TCP) zum Einsatz.
Beim STOMP-Flood Angriff werden STOMP-Pakete mit zufälligen Daten an das Zielprogramm gesendet. Dazu stellen die Bot-Rechner über den TCP-Handshake eine authentifizierte Verbindung mit dem Ziel her. Nun werden zufällige Daten in eine STOMP-Anfrage verkleidet und an den Empfänger gesendet. Durch das „Fluten“ dieser Pakete kommt es zu Einbußen der Bandbreite des Empfängers. Sollte dieser zudem versuchen die Anfragen zu parsen, kann es außerdem zu einer Überlastung des Systems kommen [43].

3.6 Infektionsrisiko

Um das Infektionsrisiko für ungeschützte IoT-Geräte, d. h. in diesem Fall Geräte mit Telnet-Zugang und Standardlogindaten, darzustellen, wurde ein Honeypot installiert. Dieser war darauf ausgelegt über eine öffentliche Telnetverbindung infiziert zu werden um Daten über die Attacke zu sammeln. Im Folgenden wird der Aufbau für den Honeypot kurz beschrieben und danach die gesammelten Daten ausgewertet.

Aufbau und Einstellungen

Beim Sammeln von Daten wurde Cowrie, ein auf SSH- und Telnet-Verbindungen ausgelegter und frei verfügbarer Honeypot benutzt. Ein Honeypot ist dabei ein System, das den Angreifer, meist durch das Vortäuschen einer Sicherheitslücke, anlocken soll, um dann Daten über die Attacke zu sammeln.
Cowrie wurde dazu auf einem RaspberryPi (im weiteren auch Honey-Pi genannt) installiert. Da das ursprüngliche Mirai-Netz zwar auch auf die Verbreitung über SSH ausgelegt ist, aber normalerweise nur Telnet genutzt wurde, wurde die SSH-Funktion des Honeypots deaktiviert und dafür die Telnet-Funktion aktiviert. Danach wurden die von Mirai benutzten Standardpasswörter als akzeptierte Passwörter in eine Liste eingetragen. Loggt sich ein Angreifer nun damit ein werden alle von ihm getätigten Eingaben aufgezeichnet. Dabei wird ein Dateisystem simuliert, auf das der Angreifer augenscheinlich Zugriff bekommt, jedoch in Wirklichkeit keine Änderungen durch-

führen kann. Da das RaspberryPi über einen Router mit dem Internet verbunden wurde, musste in diesem eine Portweiterleitung von Port 23 zum Honeypot-System eingerichtet werden. Um die gesammelten Aufzeichnungen auswerten zu können, kamen neben dem Honeypot auch noch zwei weitere Systeme zum Einsatz, die beide auf einem Virtual Private Server installiert wurden. Das erste System heißt *Modern Honey Network* (MHN) und ist für das Sammeln der Logfiles von Honeypots in einer Datenbank verantwortlich. Außerdem zeigt es nützliche Daten an, z. B. die Anzahl an Angriffsversuche, die auf dem Honeypot durchgeführt wurden. Zum detaillierten Auswerten der gesammelten Logdateien kam *Elastic Stack*, eine Softwaresammlung zum Auswerten von großen Mengen an Logdateien, zum Einsatz. Beide Systeme setzen dabei auf eine Weboberfläche.

Auswertung

Mithilfe des oben genannten Setups wurden zwischen dem 23. Mai und dem 6. Juni, also insgesamt 2 Wochen lang Daten gesammelt, welche hier analysiert werden sollen. Insgesamt wurde der Honeypot in dieser Zeit 11 161 mal angegriffen, woraus eine durchschnittliche Angriffsrate von 798 Angriffen pro Tag entsteht. Diese Zahl muss aber weiter aufgeschlüsselt werden, da es sich hier auch misslungene Logins, andere Schadsoftware sowie nicht einzuordnende Daten handelt. Wie in Abbildung 3.6.1 zu sehen waren insgesamt 83,1% der Logins erfolgreich, lediglich bei 16,9% der Versuche wurden die Logindaten abgelehnt. Hier wird schnell deutlich, was für ein hohes Sicherheitsrisiko die Standardpasswörter der Geräte mit sich bringen.

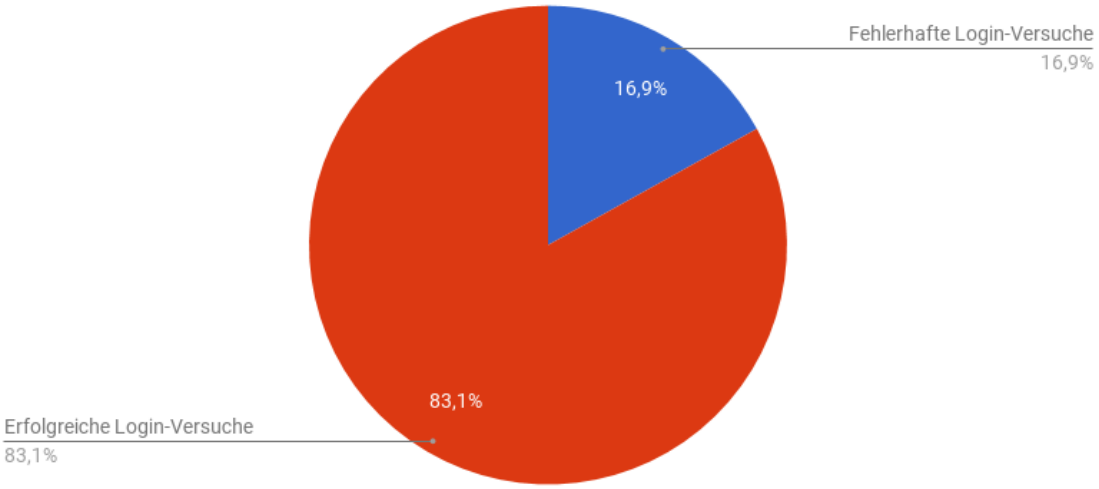

Abbildung 3.6.1: Statistik der Loginversuche auf den Honeypot: 83,1% (9 272 Versuche) waren erfolgreich, 16,9% (1 889 Versuche) waren nicht erfolgreich

Von den 9 272 erfolgreichen Angriffen können außerdem 4 240 Angriffe durch das Angriffsschema auf einen im Mai neu registrierten Trojaner namens „Linux.ProxyM.1“ zurückgeführt werden [44]. Weitere 1 491 Daten ließen sich nicht einwandfrei zuordnen, da diese meistens nur ein leeres Paket an den Honeypot schickten. Es ist zu vermuten, dass damit die offenen Ports gescannt wurden. Die restlichen 3 541 Angriffe sind auf das Mirai-Botnetz, bzw. Abwandlungen davon zurückzuführen. Dies konnte durch das Vorgehensschema vom Bot sowie vom Loader festgemacht werden. Durch die unterschiedlich genutzten Namen bei den eingegeben Befehlen ist außerdem davon auszugehen, dass es sich hier um mindestens drei verschiedene Botnetze handelt. Problematisch bei dem Testaufbau war, dass die Loader der verschiedenen Netze, welche den Bot eigentlich einmalig infizieren, immer wieder versuchten den Honeypot zu infizieren, was die vor allem die Angriffsrate verfälscht. Dies wird vorallem in Abbildung 3.6.2 deutlich. Die dort aufgeführte Liste der zehn häufigsten Angreifer besteht nämlich fast ausschließlich aus den Adressen der Load-Server.
Die Tatsache, dass die zwei häufigsten Angreifer, beide Teil eines weiterentwickelten Mirai-Bot namens „bigbotPein “, *loader* und nicht Bots sind zeigt, wie massiv die *loader* die Statistik verfälschen. Insgesamt gab es 2 306 Infizierungsversuche von *loadern*.

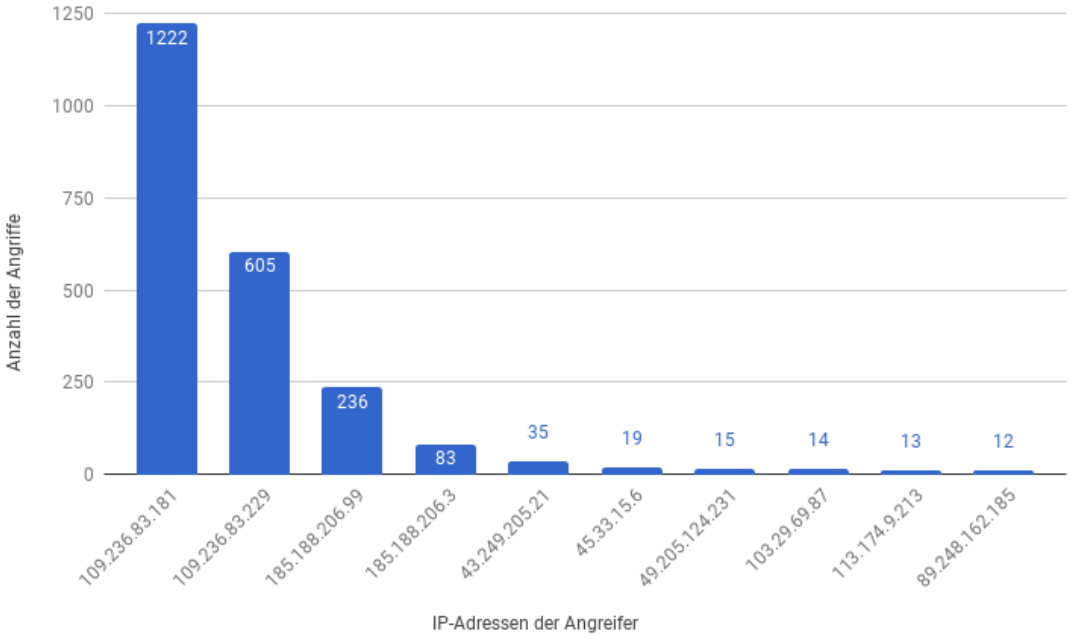

Abbildung 3.6.2: Die Angreifer, die die meisten Verbindungen mit dem Honeypot herstellten.

Es ist für eine Beurteilung des Infektionsrisikos vor allem wichtig, wie oft die Bots versuchten sich einzuloggen, denn wenn sie sich erfolgreich anmelden, ist es nur noch eine Frage der Zeit bis das System durch einen loader infiziert wird. Dies geschah 1 235 mal. Scheint diese Zahl im Gegensatz zu der Infizierungsversuchen durch den Loader noch klein, bedeutet sie dennoch eine Angriffsrate von 89 Angriffen am Tag bzw. knapp vier Angriffen in der Stunde. Sollte man also ein ungeschütztes IoT-Gerät öffentlich im Internet zugänglich machen, kann schon kurz nach der Inbetriebnahme davon ausgegangen werden, dass das Gerät mit Schadsoftware infiziert wurde.

Kapitel 4
Fazit und Ausblick

Das Ziel der Arbeit war es, den Lesern einen Überblick über Botnetze und deren Funktionsweisen zu liefern.
Für das grundlegende Verständnis von Botnetzen sind vor allem drei Punkte wichtig:

- Es gibt verschiedene Architekturen (z. B. Sternstruktur oder Peer-to-Peer-Netzwerk), die beeinflussen, wie widerstandsfähig ein Netz ist und wie schnell der Botmaster die infizierten Rechner erreichen kann. Je nach verwendetem Protokoll ist die Kommunikation zwischen den Komponenten des Botnetzes unterschiedlich komplex.

- Ein Botnetz kann auf unterschiedliche Art und Weise angreifen, sei es durch DDoS-Attacken, die ganze Server lahm legen, oder indem die Bots sensible persönliche Informationen von Zombie-Rechnern stehlen.

- Ziel eines Botnetzes ist es nahezu immer, Schaden anzurichten, deshalb müssen vom Botmaster eine Vielzahl von Verschleierungstechniken angewendet werden, um unerkannt zu bleiben. Die am besten entwickelte Methode ist hierbei das Verwenden eines Fast-Flux-Netzwerks, durch das der C&C Server nur sehr schwer zurückverfolgt werden kann.
  Umgekehrt gibt es inzwischen auch viele Erkennungstechniken, so z. B. die honeypotbasierte oder die verhaltensbasierte Erkennung.

Für das Entwickeln eines Botnetzes gibt es zahlreiche Gründe, meistens steckt jedoch die Intention, Geld zu verdienen, dahinter. So kann vom Vermieten des Netzes um Spam-Mails zu versenden über Klickbetrug auf Webseiten bis hin zum Handel mit sensiblen Personendaten auf viele Arten Profit aus einem Botnetz geschlagen werden. Es bleibt zu betonen, dass jedoch alle diese Vorgänge illegal sind.

Um nicht nur die allgemeinen Informationen über Botnetze abzubilden, wurde im zweiten Teil der Arbeit exemplarisch das Mirai-Botnetz vorgestellt. Die Wahl dieses Botnetzes stand dabei im Zusammenhang mit den aktuellen Ereignissen, vor allem dem Angriff auf DYN. Der veröffentlichte Programmcode ermöglichte es, das Botnetz und seine Funktionen relativ gut und genau zu analysieren. Neben den Grundkomponenten (C&C Struktur und Bot), die jedes Botnetz aufweist kommen beim Mirai-Botnetz zwei weitere wichtige Programme, der *loader* und der *scanlistener*, zum Einsatz, die zum Infizieren neuer Rechner benutzt werden. Dies wirkt sich auch auf die Architektur aus, bei der für jede dieser Komponenten einzelne Server verwendet werden. Bei der Analyse der Netzwerkarchitektur wurde versucht den ursprünglichen Aufbau vom Mirai-Botnetz nachzustellen, immerhin wurde das Netz inzwischen mehrfach weiterentwickelt. Während der allgemeine Lebenszyklus eines Botnetzes aus fünf Phasen besteht, kommen bei Mirai noch die Scan- und Killphase hinzu.

Mithilfe eines Honeypots wurden Daten gesammelt, um die Infektionsgefahr, die von Mirai-Botnetzen ausgehen, festzustellen. Die Daten der 11 161 Angriffe, die der Honyepot gesammelt hat, wurden dann weiter aufgeschlüsselt um zu zeigen, wie akut die Gefahr einer Infektion für ungesicherte Computer mit Telnet-Zugang ist. Die 89 Angriffe pro Tag allein von Mirai-Botnetzen machen deutlich, wie wichtig das Thema Computersicherheit selbst für vermeintlich unscheinbare IoT-Geräte ist.

Ausblick

In der Zukunft ist es wahrscheinlich, dass immer mehr größere Botnetze, vor allem im IoT-Bereich entstehen. Dies resultiert aus der stetig steigenden Anzahl an elektronischen IoT-Geräten, die zum Einsatz kommen. Laut Statistiken soll die Geräteanzahl alleine von 2016 auf 2017 um 34 %, auf 8,4 Milliarden Geräte ansteigen, 2020 sollen schon 20,4 Milliarden Geräte mit dem Internet verbunden sein [45].
Oft bewerben die Hersteller ihre Produkte als Erleichterung fürs Leben, mit der sich die Kunden um nichts mehr zu sorgen brauchen. Deshalb kann nicht erwartet werden, dass die Käufer sich von selbst um die Sicherheit all dieser Geräte kümmern. Um die Übernahme der Geräte zu verhindern, sind deshalb vor allem die Geräte-Hersteller verpflichtet für die Sicherheit ihrer angebotenen Geräte zu sorgen. Die

Fahrlässigkeit, die sich einige Hersteller in diesem Bereich leisten, z.B. durch das Setzen von Standardpasswörtern, ist dabei inakzeptabel. Dies gefährdet nicht nur den Benutzer sondern auch, wie am Beispiel des Mirai Botnetzes zu sehen, die ganze Internet-Infrastruktur. Zur Abschaffung des Problem könnte die Sicherheitspflicht der Hersteller gesetzlich geregelt werden. Ein anderer Ansatz wäre ein Zertifikat, das sich die Hersteller auf ihre Geräte ausstellen lassen können, wenn diese ausreichend gesichert sind. So könnte man gleich beim Kauf erkennen ob es sich um ein gesichertes Gerät handelt oder man den „Wolf im Schafspelz“ kauft.

Abkürzungsverzeichnis

BSI	Bundesamt für Sicherheit in der Informationstechnik
C&C	Command and Control
CDN	Content Delivery Network
DDoS	Distributed Denial of Service
DGA	Domain-Generierungs-Algorithmus
DHT	Distributed Hash Table
DNS	Domain Name System
DoS	Denial of Service
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IoT	Internet of Things
IRC	Internet Relay Chat
ISP	Internet Service Provider
P2P	Peer-to-Peer
SPOF	Single Point of Failure
TCP	Transmission Control Protocol
VPS	Virtual Private Server

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

[1] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "Botnet in ddos attacks: Trends and challenges," *IEEE Communications Surveys Tutorials*, vol. 17, pp. 2242–2270, Fourthquarter 2015.

[2] H. Gierow, "Mirai-Botnetz legte zahlreiche Webdienste lahm — golem.de." https://www.golem.de/news/ddos-massiver-angriff-auf-dyndns-beeintraechtigt-github-und-amazon-1610-123966.html, 21.10.2016. [Online; Stand 17. Mai 2017].

[3] N. Woolf, "DDoS attack that disrupted internet was largest of its kind in history, experts say — theguardian.com." https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet, 26.10.2016. [Online; Stand 17. Mai 2017].

[4] S. Hilton, "Dyn Analysis Summary Of Friday October 21 Attack — dyn.com." http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/, 26.10.2016. [Online; Stand 17. Mai 2017].

[5] B. Krebs, "Source Code for IoT Botnet 'Mirai' Released — krebsonsecurity.com." https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released/, 01.10.2016. [Online; Stand 18. Mai 2017].

[6] "Viren, Würmer, trojanische Pferde — cert.uni-stuttgart.de." http://cert.uni-stuttgart.de/themen/viren.html#malware. [Online; Stand 23. Mai 2017].

[7] Kaspersky, "Types of Malware — kaspersky.com." https://www.kaspersky.com/resource-center/threats/malware-classifications. [Online; Stand 23. Mai 2017].

[8] N. Kovac, "Malware 101: What Is Malware? — community.norton.com." https://community.norton.com/en/blogs/norton-protection-blog/malware-101-what-malware, 12.03.2015. [Online; Stand 23. Mai 2017].

[9] "Panadalabs Quarterly Report Q1 2016— pandasecurity.com." http://www.pandasecurity.com/mediacenter/src/uploads/2016/05/Pandalabs-2016-T1-EN-LR.pdf, 2016. [Online; Stand 26. Mai 2017].

[10] S. S. Silva, R. M. Silva, R. C. Pinto, and R. M. Salles, "Botnets: A survey," *Computer Networks*, vol. 57, no. 2, pp. 378–403, 2013.

[11] Y.-d. Sun and D. LI, "Overview of botnet," *Journal of Computer Applications*, vol. 7, p. 043, 2006.

[12] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, *Botnet Detection Based on Network Behavior*, pp. 1–24. Boston, MA: Springer US, 2008.

[13] M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir, "A survey of botnet technology and defenses," in *Cybersecurity Applications and Technology Conference for Homeland Security, 2009*, (Piscataway, NJ), pp. 299–304, IEEE, 2009.

[14] "Die Lage der IT-Sicherheit in Deutschland 2016 — BSI." https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2016.pdf?__blob=publicationFile&v=5, 12.03.2015. [Online; Stand 23. Mai 2017].

[15] "Global Threat Bot— www.techopedia.com." https://www.techopedia.com/definition/59/global-threat-bot-gtbot. [Online; Stand 26. Mai 2017].

[16] M. Jelasity and V. Bilicki, "Towards automated detection of peer-to-peer botnets: On the limits of local approaches.," *LEET*, vol. 9, p. 3, 2009.

[17] Matthis Laass, "Botnetze: Aufbau, Funktion & Anwendung."

[18] H. Tiirmaa-Klaar, J. Gassen, E. Gerhards-Padilla, and P. Martini, *Botnets*. SpringerBriefs in Cybersecurity, Dordrecht: Springer, 2013.

[19] J. Riden, "Know your enemy: fast-flux service networks," *The Honeynet Project*, 2008.

[20] K. Miller, "Strukturierte Peer-to-Peer Netze (Slides)." https://os.itec.kit.edu/downloads/Structured_P2P.pdf, 06.07.2009. [Online; Stand 29. Mai 2017].

[21] E. Cooke, F. Jahanian, and D. McPherson, "The zombie roundup: Understanding, detecting, and disrupting botnets.," *SRUTI*, vol. 5, pp. 6–6, 2005.

[22] N. Negash and X. Che, "An overview of modern botnets," *Information Security Journal: A Global Perspective*, vol. 24, no. 4-6, pp. 127–132, 2015.

[23] A. K. Tyagi and G. Aghila, "A wide scale survey on botnet," *International Journal of Computer Applications*, vol. 34, no. 9, pp. 9–22, 2011.

[24] "Rent a Botnet." http://www.trojaner-info.de/business-security/aktuell/rent-a-botnet.html, 27.11.2016. [Online; Stand 30. Mai 2017].

[25] A. Senpai, "Mirai Source Code— github.com." https://github.com/Philesiv/Mirai-Source-Code, 2016. [Online; Stand 09. Juni 2017].

[26] A. Senpai, "[FREE] World's Largest Net:Mirai Botnet, Client, Echo Loader, CNC source code release — hackforums.net." https://hackforums.net/showthread.php?tid=5420472&pid=52615667#pid52615667, 2016. [Online; Stand 09. Juni 2017].

[27] A. Senpai, "Forum-Post Kopie — github.com." https://github.com/Philesiv/Mirai-Source-Code/blob/master/ForumPost.md, 2016. [Online; Stand 09. Juni 2017].

[28] A. Apvrille, "IoT-based Linux/Mirai: Frequently Asked Questions - fortinet.com." https://blog.fortinet.com/2016/10/31/iot-based-linux-mirai-frequently-asked-questions, 31.10.2016. [Online; Stand 01. Juni 2017].

[29] D. Shugrue, "620+ Gbps Attack - Post Mortem - akamai.com." https://blogs.akamai.com/2016/10/620-gbps-attack-post-mortem.html, 05.10.2016. [Online; Stand 01. Juni 2017].

[30] F. Greis, "Akamai nimmt Sicherheitsforscher Krebs vom Netz — golem.de." https://www.golem.de/news/nach-ddos-attacken-akamai-nimmt-sicherheitsforscher-krebs-vom-netz-1609-123419.html, 23.09.2016. [Online; Stand 01. Juni 2017].

[31] B. Krebs, "How Google Took on Mirai, KrebsOnSecurity — krebsonsecurity.com." https://krebsonsecurity.com/2017/02/how-google-took-on-mirai-krebsonsecurity/, 03.02.2017. [Online; Stand 01. Juni 2017].

[32] D. Schirrmacher, "Rekord-DDoS-Attacke mit 1,1 Terabit pro Sekunde gesichtet — heise.de." https://www.heise.de/security/meldung/Rekord-DDoS-Attacke-mit-1-1-Terabit-pro-Sekunde-gesichtet-3336494.html, 29.09.2016. [Online; Stand 01. Juni 2017].

[33] M. Rouse, "Memory-Scraping Malware — searchsecurity.de." http://www.searchsecurity.de/definition/Memory-Scraping-Malware, 2015. [Online; Stand 04. Juni 2017].

[34] F. Lau, S. H. Rubin, M. H. Smith, and L. Trajkovic, "Distributed denial of service attacks," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 3, pp. 2275–2280 vol.3, 2000.

[35] "SYN-ACK Flood — radware.com." https://security.radware.com/ddos-knowledge-center/ddospedia/syn-ack-flood/. [Online; Stand 07. Juni 2017].

[36] "UDP Flood — incapsula.com." https://www.incapsula.com/ddos/attack-glossary/udp-flood.html. [Online; Stand 07. Juni 2017].

[37] "HTTP Flood — incapsula.com." https://www.incapsula.com/ddos/attack-glossary/http-flood.html. [Online; Stand 07. Juni 2017].

[38] "DNS Flood — incapsula.com." https://www.incapsula.com/ddos/attack-glossary/dns-flood.html. [Online; Stand 07. Juni 2017].

[39] Hemant Jain, "Mirai Botnet: Protect Your Infrastructure with FortiDDoS — fortinet.com." https://blog.fortinet.com/2016/10/24/mirai-botnet-protect-your-infrastructure-with-fortiddos, 24.10.2016. [Online; Stand 07. Juni 2017].

[40] "UDP-Based Amplification Attacks — us-cert.gov." https://www.us-cert.gov/ncas/alerts/TA14-017A, 04.11.2016. [Online; Stand 07. Juni 2017].

[41] "Generic Routing Encapsulation — wikipedia.org." https://de.wikipedia.org/w/index.php?title=Generic_Routing_Encapsulation&oldid=165062231, 2017. [Online; Stand 07. Juni 2017].

[42] F5 DevCentral, "Mirai Botnet and GRE Floods — youtube.com." https://www.youtube.com/watch?v=qaZL6nwU3h8, 22.02.2017. [Online; Stand 12. Juni 2017].

[43] Dima Bekerman and Dan Breslaw, “How Mirai Uses STOMP Protocol to Launch DDoS Attacks — incapsula.com.” https://www.incapsula.com/blog/mirai-stomp-protocol-ddos.html, 15.11.2016. [Online; Stand 07. Juni 2017].

[44] “Linux.ProxyM.1 — drweb.com.” https://vms.drweb.com/virus/?i=15389229&lng=en, 2017. [Online; Stand 11. Juni 2017].

[45] F5 DevCentral, “Gartner Says 8.4 Billion Connected "Things"Will Be in Use in 2017, Up 31 Percent From 2016 — gartner.com.” http://www.gartner.com/newsroom/id/3598917, 07.02.2017. [Online; Stand 12. Juni 2017].

Anhang

Inhalt des Kapitels

A Verwendete Standardpasswörter von Mirai

Benutzer	Passwort
root	xc3511
root	vizxv
root	admin
admin	admin
root	888888
root	xmhdipc
root	default
root	juantech
root	123456
root	54321
support	support
root	(none)
admin	password
root	root
root	12345
user	user
admin	(none)
root	pass
admin	admin123
root	1111
admin	smcadmin

Benutzer	Passwort
admin	1111
root	666666
root	password
root	1234
root	klv123
Administrator	admin
service	service
supervisor	supervisor
guest	guest
guest	12345
guest	12345
admin1	password
administrator	1234
666666	666666
888888	888888
ubnt	ubnt
root	klv1234
root	Zte521
root	jvbzd
root	anko
root	zlxx

Benutzer	**Passwort**
root	7ujMko0vizxv
root	7ujMko0admin
root	system
root	ikwb
root	dreambox
root	user
root	realtek
root	00000000
admin	1111111
admin	1234
admin	12345
admin	54321
admin	123456
admin	7ujMko0admin
admin	4321
admin	pass
admin	meinsm
tech	tech
mother	fucker

B Ausgeschlossene IP-Adressen des Mirai-Bots

Adressbereich	Zuordnung
127.0.0.0/8	Loopback-Adresse
0.0.0.0/8	Ungültiger Adressraum
3.0.0.0/8	General Electric
15.0.0.0/7	Hewlett-Packard (HP)
56.0.0.0/8	United States Postal Service
10.0.0.0/8	Internes Netzwerk
192.168.0.0/16	Internes Netzwerk
172.16.0.0/14	Internes Netzwerk
100.64.0.0/10	IANA NAT reserved
169.254.0.0/16	IANA NAT reserved
198.18.0.0/150	IANA Special use
224.*.*.*+	Multicast
6.0.0.0/8	US Verteidungsministerium
7.0.0.0/8	US Verteidungsministerium
11.0.0.0/8	US Verteidungsministerium
21.0.0.0/8	US Verteidungsministerium
22.0.0.0/8	US Verteidungsministerium
26.0.0.0/8	US Verteidungsministerium
28.0.0.0/8	US Verteidungsministerium
29.0.0.0/8	US Verteidungsministerium
30.0.0.0/8	US Verteidungsministerium
33.0.0.0/8	US Verteidungsministerium
55.0.0.0/8	US Verteidungsministerium
214.0.0.0/8	US Verteidungsministerium
215.0.0.0/8	US Verteidungsministerium

C SQL-Code zum Erstellen der nötigen MySQL-Datenbank

```
CREATE DATABASE mirai;

CREATE TABLE `history` (
`id` int(10) unsigned NOT NULL AUTO_INCREMENT,
`user_id` int(10) unsigned NOT NULL,
`time_sent` int(10) unsigned NOT NULL,
`duration` int(10) unsigned NOT NULL,
`command` text NOT NULL,
`max_bots` int(11) DEFAULT '-1',
PRIMARY KEY (`id`),
KEY `user_id` (`user_id`)
);

CREATE TABLE `users` (
`id` int(10) unsigned NOT NULL AUTO_INCREMENT,
`username` varchar(32) NOT NULL,
`password` varchar(32) NOT NULL,
`duration_limit` int(10) unsigned DEFAULT NULL,
`cooldown` int(10) unsigned NOT NULL,
`wrc` int(10) unsigned DEFAULT NULL,
`last_paid` int(10) unsigned NOT NULL,
`max_bots` int(11) DEFAULT '-1',
`admin` int(10) unsigned DEFAULT '0',
`intvl` int(10) unsigned DEFAULT '30',
`api_key` text,
PRIMARY KEY (`id`),
KEY `username` (`username`)
);
```

```
CREATE TABLE `whitelist` (
`id` int(10) unsigned NOT NULL AUTO_INCREMENT,
`prefix` varchar(16) DEFAULT NULL,
`netmask` tinyint(3) unsigned DEFAULT NULL,
PRIMARY KEY (`id`),
KEY `prefix` (`prefix`)
);
```

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ort, Datum	Philipp Wenskus