



University of
Zurich^{UZH}

Botnet Command-and-Control Traffic Analysis

Getoar Galloopeni
Zurich, Switzerland
Student ID: 16-727-125

Supervisor: Bruno Rodrigues, Muriel Franco
Date of Submission: January 30, 2020

Abstract

Vernetzte Geräte in privaten Haushalten setzen den Fokus oft auf Komfort und werden in puncto Sicherheit und Risiko nicht Computern gleichgesetzt. Kleingeräte im Haushalt eignen sich besonders als Mittel für zentral gesteuerte, gross angelegte Distributed Denial-of-Service (DDoS) Attacken. Das hatte zur Folge, dass in den Jahren 2016 und 2017 viele Attacken mithilfe eines Botnets namens Mirai durchgeführt werden konnten [8].

Diese Arbeit versucht, nach erfolgreicher Implementation von Mirai, anhand dessen Funktionsweise ein Erkennungssystem zu entwickeln. Dabei liegt der Fokus auf der Analyse des Datenflusses vom oder zum sogenannten Command-and-Control (CnC) Server. Beschrieben wird zuerst die Implementation eines Prototypen zur Analyse von Packet Capture (PCAP) Dateien. Die Erkenntnisse über Mirai helfen, Angriffe mit minimaler Latenzzeit zu identifizieren. Das ist wiederum entscheidend für das automatisierte Bekämpfen solcher Attacken. Die Effektivität von Echtzeiterkennung wird getestet und die Resultate evaluiert.

Nach der Entwicklung von erfolgreichen Erkennungsmechanismen werden schliesslich geeignete Netzwerk-Tools, wie Firewalls oder Queues, eingesetzt, um die Angriffe abzuwehren. Zuletzt wird die Wirksamkeit der Massnahmen zur Mitigierung solcher Angriffe geprüft und deren Resultate evaluiert.

Smart devices in private households often seek convenience and are not treated equal to personal computers in regards of security and risk. Small devices in the household are particularly suitable as a means for centrally controlled, large-scale Distributed Denial-of-Service (DDoS) attacks. As a result, in 2016 and 2017 many attacks could be carried out using a particular botnet called Mirai [8].

This thesis tries, after the successful implementation of Mirai, to develop a detection system based on its functioning. The focus lies on the analysis of the data flow coming from or going to the so-called *Command-and-Control (CnC)* server. First, the implementation of a prototype for the analysis of *Packet Capture (PCAP)* files to detect Mirai activity is described. The knowledge of Mirai will then help to perform live detection with minimal latency. This in turn is crucial for the automated counteraction of such attacks. The effectiveness of real-time detection is tested and the results are evaluated.

After developing successful detection mechanisms, tools such as firewalls or queues are used to defend against the attacks. Finally, the effectiveness of the mitigation measures is tested and the results are evaluated.

Acknowledgments

I am using this occasion to thank my supervisors Bruno Rodrigues, Muriel Franco and Prof. Dr. Burkhard Stiller for their support throughout the thesis. A special thanks goes to Bruno Rodrigues for always being available for all kinds of questions and support.

Last but not least, I thank my family and friends for supporting me to pursue my thesis and motivating me along the way. Especially, I want to thank those who took the time to proof read this work.

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
1.1 Motivation	1
1.2 Description of Work	2
1.3 Thesis Outline	3
2 Background	5
2.1 Botnets	5
2.2 Distributed-Denial-of-Service attacks	6
2.3 Network Intrusion	8
3 Related Work	9
3.1 Botnet monitoring	9
3.2 Bot Scanner Summation Prototype	10
3.3 Network monitoring and early bot detection	11
3.4 Botnet probing	13

4	Design	15
4.1	Mirai	15
4.2	Testbed scenario	16
4.3	Traffic Monitoring	18
4.4	Defensive countermeasures	18
4.5	Implementation approach overview	19
5	Implementation	21
5.1	Mirai	21
5.1.1	Bug Fixes	21
5.1.2	Database	22
5.1.3	CnC Server	23
5.1.4	Custom Encoding	24
5.1.5	Bots	24
5.2	DNS Server	26
5.3	Host disinfection	26
5.4	Retrospective Analysis (PCAP Anaylsis)	27
5.5	Real-time analysis	30
5.6	Automated implementation of mitigating measures	31
5.6.1	Queues	32
5.6.2	Firewall filters	33
6	Evaluation	35
6.1	Evaluation Setup	35
6.2	Retrospective Detection	36
6.2.1	Key Performance Indicators and Thresholds	36
6.2.2	Scenarios	37
6.3	Real-time detection	38
6.4	Mitigation Measures	40

<i>CONTENTS</i>	vii
7 Conclusion	43
7.1 Summary and conclusion	43
7.2 Future work	44
Bibliography	44
Abbreviations	47
List of Figures	47
List of Tables	49
List of Listings	51
A Usage guidelines	55
A.1 Retrospective analysis	55
A.2 Real-time Analysis	56
B Contents of the CD	59

Chapter 1

Introduction

1.1 Motivation

As internet access becomes more progressive, including a broad set of users using a variety of devices, there is the concern about device (in)security and awareness of personal security (*i.e.* privacy). In addition to the growing number of connected devices, there is the increase of available bandwidth allowing these devices to exchange a greater volume of data in support of current services and experiences such as online gaming and video streaming. In this regard, the progressive improvement of accessibility and democratization of the cyberspace also enables its malicious usage, being able to compromise not only users or devices but entire societies.

In its 2016-2017 Cyber threat report, the United Kingdom National Crime Agency named Distributed-Denial-of-Service (DDoS) attacks as the biggest cyber threat, counting examples from the Ukrainian Power Supply, the US Democratic Party and the Bangladesh Bank having suffered from this type of attack [1]. Furthermore, in 2017, the WannaCry ransomware infected more than 230'000 computers in over 150 countries, affecting hospitals, universities, companies and government organizations [5]. Denial-of-Service (DoS) attacks are known for many years, but the constantly evolving context of DDoS attacks makes them more relevant and feared than ever. This goes back to the increasing number of smart devices in our everyday life, which are also referred to as Internet-of-Things (IoT) devices. When in 2018 there were approximately 7 billion smart devices estimated, the forecast for 2025 stated approximately 21.5 billion devices, making an increase of 307% [18].

The increasing concerns about cyber attacks compel organizations to delay the adoption of technologies (such as Cloud and IoT) until they are proven to be mature enough security-wise. However, there is a continuous trade-off between keeping pace with novel technologies and risk mitigation strategies. As the former is usually followed by innovative startup tech companies, the latter is typically the approach defined by companies long established in the market. Specifically, DDoS attacks are one of the major current Internet issues, being a major threat to the availability of web services.

Behind such attacks are adversaries who systematically exploit vulnerabilities to infect thousands of machines with malware. *Malware* is a broad description for a software that is harmful to the systems' users and was developed to perform undesirable or malicious activities on the system [11]. Botnets are frequently used for profit-driven crime and an underground economy has formed around them [31], which has been responsible for major economical damage. In order to tackle the problem that botnets pose, an understanding of both the different parties involved in the defense from botnet-related attacks as well as the attackers and adversaries behind the botnets is necessary.

Since smart devices in private households often seek convenience and are not treated equally impactful as personal computers, their potential security issues are also ignored. A malware which became famous for exploiting these weaknesses is called *Mirai* [3]. It is a malware which scans through ranges of Internet Protocol (IP) addresses looking for devices to attack. Once found, a set of default credentials is used in order to log in and take control over the device. This procedure is executed repeatedly until the attacker has set up a network of controlled devices, a so-called *botnet* [31].

A comparable set up with a botnet size of more than 500'000 bots, has been used for attacks on network providers OVH, TalkTalk, Dyn, Twitter and others [1]. Since the source code of Mirai has been published by the author, everyone can use it. Mirai is a dangerous malware but eventually, it is the framework used to exploit another weakness which is the main problem, namely the unchanged default credentials. There are already variations of Mirai, for instance Okiru, Satori, Masuta and others, which make use of different weaknesses of smart devices [8, 28].

1.2 Description of Work

This thesis demands, at an initial stage, to acquire the conceptual elements as well as to understand the baseline code of Mirai in order to fulfill the thesis objectives. Also, initial steps include understanding the overall problem to be tackled by the thesis and possible requirements through the analysis of related work. The expected outcome is a sufficient

understanding of the background to accomplish the thesis objectives and obtain a holistic overview of required activities in order to list and organize the activities in a timeline.

The second stage involves more research-oriented work with respect to the analysis of the control flow channel between the botmaster and bots in order to extract communication patterns. Once the first thesis objective is achieved, a baseline for the Mirai CnC traffic analysis will be created by simulating an attack and the extracting of traffic data. Driven by the description of the outlined work description, the goals just mentioned are to be achieved.

1.3 Thesis Outline

The thesis is structured as follows. In Chapter 2, the theoretical background of Mirai and network monitoring in general is described. Chapter 3 provides insights into prior related work in the area of botnet monitoring and botnet detection. In Chapter 4, the technical design is presented. This reaches from the experimental setup used for the thesis up to the analysis methodology and approach. Chapter 5 provides the actual implementation of the traffic analysis along with preceding assumptions and necessary adaptations of the Mirai software code. In Chapter 6, the analysis of the preceding Chapter is evaluated. Ultimately, the thesis is concluded by Chapter 7 which also contains information about future work in the area of CnC traffic analysis.

Chapter 2

Background

2.1 Botnets

A botnet is a clustering of connected devices controlled by at least one attacker, *i.e.*, the so called botmaster [32]. The term *botnet* is derived from the two words *robot* and *network* and refers to a network of robot-like devices. The relation between controlling and controlled devices is often called *master-slave-relation*, which leads to the term *botmaster* representing the person. Although infected devices are typically a consequence of severe security breaches, they are mostly used as a means to an end, such as phishing or DDoS attacks [19]. Botnets can reach proportions of up to 400'000 controlled devices and have reportedly been offered for rent in a 'DDoS-as-a-Service' type of attack, also known as *booters* [14]. To compose a network of bots, the attacker can subsequently scan IP address ranges for possible victims and infect them with malicious software which lets the botmasters gain control [14]. Because of their insidious manner, it is difficult to identify a botnet or individual infected devices. Botnets often do not harm the bot owners, can have long inactive phases and are able to hide their data inside other applications' traffic [14].

As described by Barford et al. [4], in most cases, botnet architectures make use of communication protocols. Some of these protocols may be the Internet-Relay-Chat (IRC), Hypertext-Transfer-Protocol (HTTP), the Post-Office-Protocol (POP) or a Peer-to-Peer (P2P) system [19].

The *CnC server* is the leading instrument to communicate with the bots [19] and is the main differentiating component between botnets and other malware [10]. Through the

CnC server, the botmaster can launch DDoS attacks, e-mail spam attacks, distribute spyware and much more [19]. The communication for such attacks could be performed through the Tor network [14]. The *reporting server* is considered the component which is contacted after a new host has been infected. The *loader*, also called *loading server*, is the component triggered after the reporting server has registered the infected host, i.e. the new *bot*. The loader disseminates host devices with host-specific part of the botnet malware (the bot module), built and compiled for 18 different platform architectures, including ARM, MIPS and x86 [3]. When loaded, the infected host device is then ready to receive and execute remote commands.

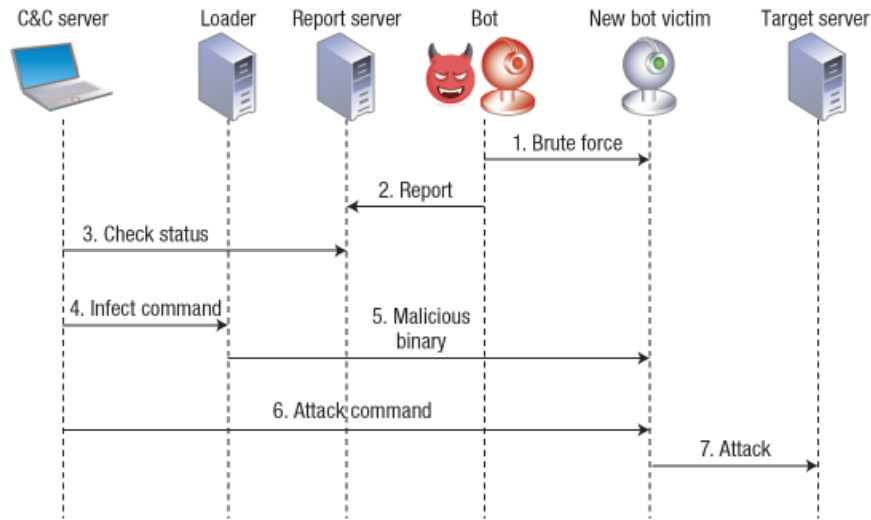


Figure 2.1: Botnet architecture including basic communication and operation [14]

For botnet sizes such as the one used in the attack of the DNS company Dyn [12], the prior mentioned component occurred redundantly [19]. Consequently, attackers were able to control hundreds of thousands of infected devices, scaling the spreading process by redundant loaders and reporting servers.

2.2 Distributed-Denial-of-Service attacks

A DDoS attack consists of multiple, often compromised, devices that are used to target other systems, most of them being web servers. This type of attack tries to overwhelm computer systems so that they cannot provide regular service to other, legitimate users [2].

DDoS attacks are, in most respects, similar to DoS attacks. While in DoS attacks the requests derive from an individual host, in DDoS attacks the attacker(s) strike from many,

distributed hosts. Depending on the relationship between the attacker and the attacking devices, the first part of the attack could consist of infecting as many nodes as possible. In the age of virtualization, it is also possible to set up a self-owned botnet. In order to perform the large-scale attacks, like the ones of the last years [12], this strategy would be too uneconomical.

Figure 2.2 gives an overview over a basic DDoS architecture. This type of system composition is also known as the *Agent-Handler-Model* [30], where in this case the handlers are the CnC servers and the agents are the infected hosts.

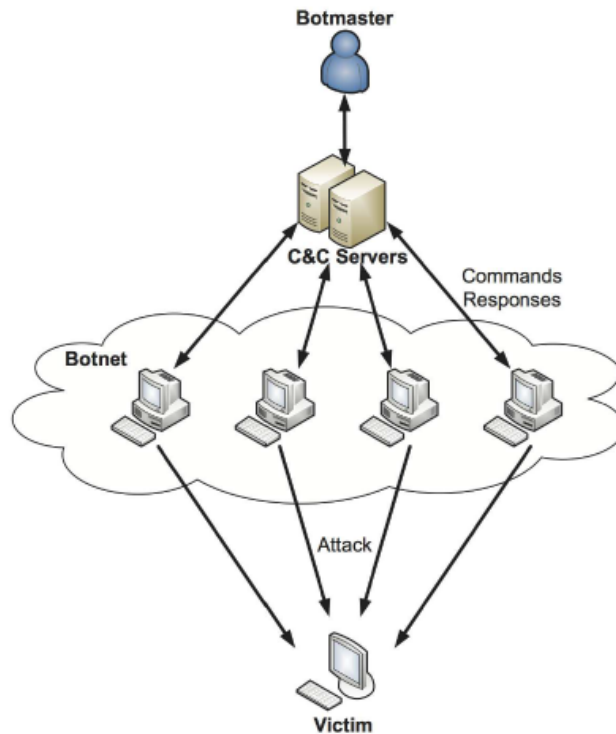


Figure 2.2: Basic DDoS architecture resulting from a botnet [19]

There are different kinds of DDoS attacks. Most of them target a computer network's bandwidth or connectivity and are referred to as *volume-based attacks* [6]. Other attack types are Operating System (OS) level attacks where the way an OS implements a protocol is taken advantage of. Furthermore, DDoS attacks can be classified not only by the type of exploited vulnerability, but also by the degree of automation, the attack rate dynamics and the resulting impact [6].

2.3 Network Intrusion

In the age of digitization and continuous connectivity, the increasing number of connected devices is followed by a higher risk. When it comes to maintaining the network's integrity, one differs between *Intrusion Detection Systems (IDS)* and *Intrusion Prevention Systems (IPS)*, sometimes also referred to as one system *IDPS* [13]. Among IDS, there are *Host Intrusion Detection Systems (HIDS)*, which often is adopted by Anti-Virus systems, and *Network Intrusion Detection Systems (NIDS)* [13]. A NIDS is able to monitor data packets moving within a network or, upon configuration, monitor a subset of a network [13]. Also, it may selectively scan packets upon specified rules or procedures. Essentially, the intrusion detection methodologies are classified into three categories: Signature-based detection, anomaly-based detection and stateful protocol analysis - each with its own benefits and drawbacks [16].

Signature-based detection	
<u>Pros</u>	<u>Cons</u>
• Simplest method • Most efficient method • Detailed contextual analysis	• Ineffective for yet unknown attacks • Little understanding of states and protocols • Keeping signature database up to date requires high effort • Maintaining the knowledge is time-consuming
Anomaly-based detection	
<u>Pros</u>	<u>Cons</u>
• Effective for new vulnerabilities • Less dependent on OS • Facilitates detections of privilege abuse	• Unavailable during rebuilding of behavior profiles • Difficult to trigger alerts in right time • Weak profile accuracy since observed events constantly changes

Table 2.1: Comparison of the pros and cons of the signature-based and anomaly-based detection methods by Liao, Hung-Jen, et al. 2013 [16]

The method which is mostly used is the signature-based detection method [16]. It is the most simple and also the most effective method when it comes to detecting known attacks, but demands high actuality of the signature database and continuous updates which is costly [16]. Its most considerable drawback is the detection of unregistered attacks where no signatures exist yet. Anomaly-based detection methods which focus on suspicious activities in the network, contrarily, are able to detect attacks which have never been seen before.

Chapter 3

Related Work

3.1 Botnet monitoring

After the first Mirai attacks in 2016, the authors of the paper *Understanding the Mirai Botnet* [3] started a research. Participants from Akamai Technologies, Cloudflare, Google, Georgia Institute of Technology and others were involved in the analysis. They inspected the growth of multiple botnet clusters in a seven-month retrospective analysis of Mirai’s development until it reached the size of 600’000 bots [3]. The work does not only focus on the botnet growth, but it also does highlight its composition, evolution and DDoS activities in particular attacks. The device types, models and vendors could be identified for approximately 587’000 devices out of 1.8 million active scan banners [3]. A breakdown of these insights is shown in Table 3.1.

CWMP (28.30%)		Telnet (26.44%)		HTTP (19.13%)		FTP (17.82%)		SSH (8.31%)	
Huawei	3.6%	Dahua	9.1%	Dahua	36.4%	D-Link	37.9%	MikroTik	3.4%
ZTE	1.0%	ZTE	6.7%	MultiTech	26.8%	MikroTik	2.5%		
		Phicomm	1.2%	ZTE	4.3%	ipTIME	1.3%		
				ZyXEL	2.9%				
				Huawei	1.6%				
Other	2.3%	Other	3.3%	Other	7.3%	Other	3.8%	Other	1.8%
Unknown	93.1%	Unknown	79.6%	Unknown	20.6%	Unknown	54.8%	Unknown	94.8%

Table 3.1: The most infected device vendors as a fraction of Mirai banners found by Censys [3]

Summarizing, the investigation made it possible to gather detailed insights into the context of Mirai attacks during 2016 and 2017. As depicted in Table 3.1, the study also includes the top 14 targets of the observation period.

Target	#Attacks	Notes
Lonestar Cell	616	Liberian telecom targeted by 102 reflection attacks.
Sky Network	318	Brazilian Minecraft servers hosted in Psychz Networks data centers.
1.1.1.1	236	Test endpoint. Subject to all attack types.
104.85.165.1	236	Unknown router in Akamai's AS.
feseli.com	157	Russian cooking blog.
minomortaruolo.it	157	Italian politician site.
Voxality hosted C2	106	C2 domain from DNS expansion.
Tuidang websites	100	HTTP attacks on two Chinese political dissidence sites.
execrypt.com	96	Binary obfuscation service.
auktionenhilfe.com	85	Russian auction site.
houtai.longqikeyi.com	85	SYN attacks on a former game commerce site.
Runescape	73	World 26 of a popular online game.
184.84.240.54	72	Unknown target hosted at Akamai.
antiddos.solutions	71	AntiDDoS service offered at react.su.

Table 3.2: The top Mirai victims in a seven-month observation period

3.2 Bot Scanner Summation Prototype

The *Bot Scanner Summation Prototype* [8], here called *Summation Prototype*, is the product of the doctoral thesis of Charles V. Frank Jr. It is a script written in Python to tabulate the amount of infected Mirai bots, potential victims and the amount of network packet types which can be associated with another bot attempting to connect to a new bot victim [8].

For the analysis, a dataset containing bot scanning activity of more than six months has been used [8]. For each day of bot scanning activity, one PCAP file was created [8]. The *Summation Prototype* linearly scans the dataset for the IP address of the scanning bot in order to calculate the number of bots which are already infected [8]. A similar procedure is used to identify all potential victims. While scanning the same dataset, the destination IP addresses packets sent from a scanning bot are interpreted as potential new bot victims [8]. In Figure 3.3, an example of a summation report is shown. It contains key data of summation time range, the scanned subnets, summation of different kind of packets as well as aggregated data like the average number of packets per minute [8].

The *Summation Prototype* is a useful tool for analyzing structured network data and extracting reporting-like information. However, some of the gained insights are limited in their expressiveness. As Mirai generates IP addresses, which it tries to connect to [29], not every IP address automatically represents a host that could be a possible victim. This is referred to as *brute forcing attacks* [14]. As an example, assuming the router of a private network with several infected devices starts scanning for possible victims. By

scanning only the class B IP address range (192.0.0.0/16), a maximum of 65'536 [17] could be identified as possible bot victims which is not in proportion to the actual number of devices.

Summary for bot scanning dataset	
Start Date	2016-12-20
End Date	2016-12-24
Destination Subnets	['199.9.14.0/24', '192.228.79.0/24', '130.152.184.0/24']
Total number of packets	2'702'090
Total number of successful SYN packets	1'242'849
Total number of re-transmission packets	1'459'241
Average number of bot scanning	118'556.40 (per PCAP)
Average number of potential new bot victims	513.60 (per PCAP)
Average number of packets	375.29 (per minute)
Average number of bots scanning	82.33 (per minute)
Average potential new bot victims	0.36 (per minute)

Table 3.3: Dataset summary from the *Bot Scanner Summation Prototype* for a selected date range

3.3 Network monitoring and early bot detection

As Mirai's spreading mechanism needs to run for weeks and months to reach the size of a high profile attack as OVH and Dyn experienced [3, 14]. There are academic and industrial attempts for efficient early detection solutions which could help stem the risk of an attack. In this context, Kumar, Ayush, and Teng Joon Lim (2019) [15] developed a bot detection algorithm following a two-dimensional (2D) packet sampling approach. Specifically, traffic data was sampled both across different devices in a network and over time. The identified characteristics were then leveraged, developing the proposed detection algorithm [15].

In a dedicated testbed, an IoT (Internet-of-Things) environment was emulated and Mirai's scanning traffic was captured [15]. After installing and running Mirai, a total of over 1.5 million packets could be captured for further analysis. From this data, all scanning-related packets could be determined as SYN (synchronization) packets. 90% of those aim destination port 23 (*i.e.*, Telnet) and the remaining 10% aim port number 2323, both being Telnet ports [15]. Additionally, occasional *keep alive traffic* was observed, consisting of PSH (push) + ACK (acknowledge) packets [15].

Furthermore, another dataset was collected without any Mirai traffic. In a testbed without Mirai scanning processes running, over 10'000 packets were captured and stored [15]. This data was used as a reference and represented the network behavior in a safe environment.

The evaluation of the latter has indicated that not a single SYN packet with target port 23 or 2323 was captured in the simulation without Mirai activity [15]. This strengthens the assumption that any SYN traffic between IoT devices is a strong indication of botnet activity.

Even though the detection process sounds viable, an inspection of the TCP header of every packet in the network could be too expensive in terms of computational resources [15]. Assuming every IoT device is powered as soon as it is installed, a calculation estimated the total data rate peak targeting one major ISP (Internet Service Provider) in the U.S. to be around 14,895 billion bits/s [15]. Therefore, the suggested approach is to sample only a fraction of the traffic. This can be formulated as an optimization problem with the objective to minimize the costs associated with a *delayed detection*, while maximizing the *detection rate* [15]. Considering other relevant parameters as *sampling frequency* and the ratio between *vulnerable and non-vulnerable devices*, a formal optimization problem was proposed, leading to a detection algorithm being tested for efficiency and effectiveness [15]. As shown in Figure 3.1, the detection delay drops with increasing sampling frequency. The testbed included 100 IoT devices and a vulnerable / non-vulnerable device ratio of 0.4, such that 100k packets per device could be captured [15].

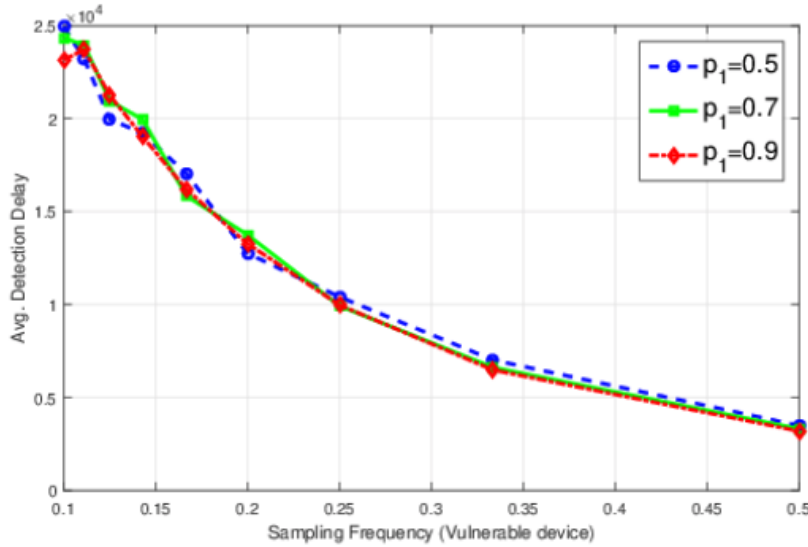


Figure 3.1: Trade-off between expensive sampling frequency and impactful detection delay for attack probabilities $p_1 = 0.5, 0.7, 0.9$ [15]

The detection algorithm proposed and evaluated by Kumar, A., & Lim, T. J. (2019) proves the feasibility of effective detection algorithms. Considering the long growing process of a botnet, a certain detection delay is reasonable such that the proposed implementation could possibly be realized economically. If the required investment is considered valuable by the ISPs, remains to question.

3.4 Botnet probing

Once malicious bot traffic is suspected, there are various methods to validate the hypothesis. Gu, Guofei, et al. (2009) presented an algorithmic framework which should help separating botnet dialogues from human conversations. The approach followed is to actively manipulate selected, suspicious sessions to identify botnet dialogues rather than passively monitoring the network flow [11]. However, some of the presented probing techniques are not applicable for modern botnets like Mirai.

Chapter 4

Design

The following Chapter contains the design of existing software such as the original release of Mirai on GitHub as well as design rationales of the experimental setup used for this thesis. Furthermore, the imitation of an own Domain Name System (DNS) server and the collection of traffic data are outlined.

4.1 Mirai

Mirai, which is a Japanese word meaning "the future", is a malware used to create large-scale botnets in a highly automated manner [14]. In September 2016, it drew public attention after being used to attack the popular security website Krebs On Security [29]. Afterwards, a larger botnet of about 400'000 bots [1] attacked the french hosting company OVH, which reportedly peaked at an inbound traffic of 1 Tbit/s [29].

One of Mirai's strengths is its distinct, worm-like spreading mechanism. When active, Mirai continuously generates IP addresses in order to connect to them via SSH or Telnet. It uses an included set of 62 default username and password combinations [29]. Whenever a successful connection is established, the IP address of the new victim is sent to a reporting server along with the successful credentials. The new victim receives the malware from the loader and continues the spreading mechanism until an attack is started [14].

From an architectural perspective, the initial variant of Mirai, described by Sinanovic and Mrdovic (2017) [29], has no significant differences from the general botnet architecture. However, there are variations of Mirai with compelling disparity in their composition, which is described in more detail by Antonakakis, Manos, et al. 2017 [3] and by Frank 2019 [8]. For this thesis, the initially published version of Mirai will be used which has been made available by Jerry Gamblin 2017 [9].

Mirai surprised at the beginning with its simplicity. Unlike complex P2P systems, the behavior of the default Mirai botnet is fairly comprehensible by analysing its code components. Sinanovic and Mrdovic (2017) [29] performed a comprehensive analysis of the Mirai code. The authors provided a low-level description of the Mirai segments, which is the foundation of how Mirai works and interacts. The paper covers all static aspects and demonstrates the included SQL table structures. In Table 4.1, one of the database's tables is presented. The HISTORY table keeps track of the CnC history within an instance of a botnet.

Field	Type	Key	Default
user_id	int(11)	MUL	NULL
time_sent	timestamp	-	TIMESTAMP
duration	int(11)	-	NULL
command	varchar(500)	-	NULL
max_bots	int(11)	-	NULL

Table 4.1: Mirai's HISTORY table structure including fieldnames, types, keys and default values from Sinanovic and Mrdovic (2017) [29]

After a command has been launched, the malware inserts a record into this table. The stored information are, for instance, the identifier (ID) of the user sending the command, a timestamp from the command time, the duration of the command execution and the command itself which is an interpretation of a string [29].

The detection in Sinanovic and Mrdovic (2017) bases solely on the port numbers, which can be identified during the code analysis. Mirai's most popular variants communicate over Telnet, HTTP or SSH, and therefore target ports like 22, 23 or 80 [3, 29]. While this may work when implemented as an IDS rule, a static detection method could fail after slight modifications of the source code and make botnet activity undetectable.

4.2 Testbed scenario

To imitate the environment and components where Mirai was operated on, a setup was created as shown in Figure 4.1. The goal was to establish a setting as realistic as possible.

The main difference of analyzing traffic in the presented setup instead of real-world traffic from an ISP is the significantly lower amount of unrelated traffic. This kind of traffic is often referred to as *noise*. The missing noise of the experimental setup could therefore have simplifying effects on the pattern analysis process.

As shown in Figure 4.1, the setup consists of six physically independent hosts, a switch, a router and a control terminal. The hosts, which imitate vulnerable IoT devices like cameras or baby phones, are Asus tinker boards with quad-core 1.6 GHz processors and 2 GB of RAM each. The tinker boards run TinkerOS Debian 9 v2.0.11. The traffic is switched through a D-Link DGS-105 to a MikroTik RouterBoard hex series running v6.42.4 [22]. The control terminal is a HP ProBook with a quad-core i5 @2.67 GHz and 4 GB of RAM running Linux Ubuntu 18.04.

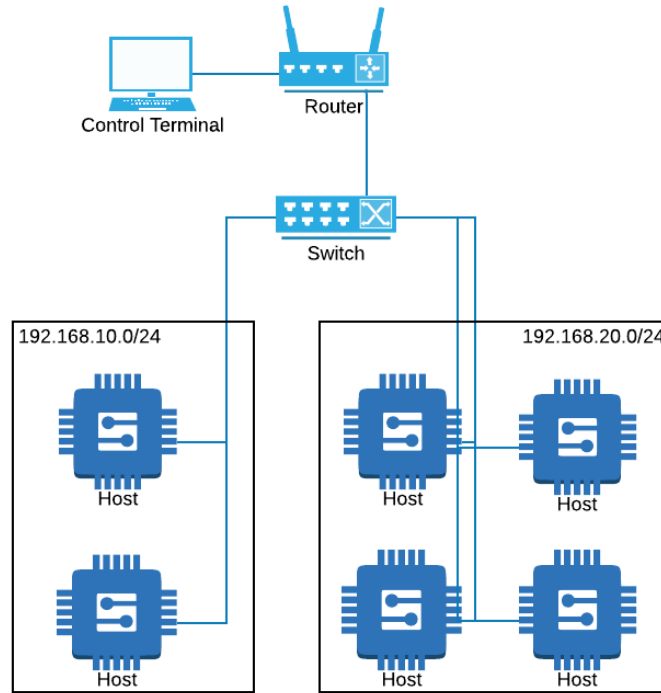


Figure 4.1: Overview of the experimental setup including all involved components

Scanning, controlling and attacking were performed in two private networks with sub-masks 192.168.10.0/24 and 192.168.20.0/24 without any connection to the internet. This was a risk mitigation measure not to scan or attack public devices or consume the bandwidth of public networks. The control terminal, in the context of a Mirai botnet [29], served as CnC server, as a reporting server and also as a loader. The connection from the infected hosts to the CnC server is established by resolving a domain name to an IP address [8].

4.3 Traffic Monitoring

The data generated by scanning, infecting, connecting, reporting and attacking is subject to the analysis. The data is analyzed using *Wireshark* [33], a popular and widely used network analysis tool. Wireshark runs on the control terminal as shown in Section 4.2. In order for the control terminal to analyze all packets, a forwarding rule is implemented using the router's packet sniffing feature [21]. The router then forwards all packets of other interfaces to a specified IP address with destination port 37008 [7]. By configuring Wireshark to listen to UDP port 37008 as described in [7], the packets of the network can be fully analyzed.

4.4 Defensive countermeasures

Once Mirai activity is assumed or even confirmed, various countermeasures can be taken on different levels. One differentiates between measures which can be taken either on the victim or on the network.

Victim. There are few measures which could be taken in order to protect the victim by itself. Mirai offers a variety of attack vectors which include choosing of parameters such as altering target ports or sequence numbers. Therefore, one of the few counteractions is closing the interface for the attack duration. This countermeasure has large drawbacks and would therefore not be suitable for productive devices. Particularly, the disconnected host could not do any service requiring a connection which equals the ultimate goal of a (D)DOS attack, namely to impair the availability of the device. Also, the network would still suffer from the attack which, in some circumstances, exhausts a majority of the bandwidth.

Network. The network is the more appropriate level to take measures on. It is very difficult to detect malicious activity in the first place, especially as botnets like Mirai are designed not to be detected. But once malicious activity is assumed, there are steps which can be taken without interfering with business-as-usual. It is necessary to differentiate between the stages where suspicious network activity has to be further analysed and when Mirai command-and-control activity could clearly be detected. In the first case, there are various network shaping possibilities one could take. Queues offer a large functionality to restrict traffic upon certain criteria. For the latter, firewall rules are the most effective measure for defense.

4.5 Implementation approach overview

The implemented scripts of this thesis are partly isolated and meet stand-alone obligations. Individually, each part causes a counteraction, sometimes even an elimination of Mirai. However, since they belong in a coherent system, their affiliations and assignments are shown in Figure 4.2. Red markings in the Figure indicate a part of the implementation.

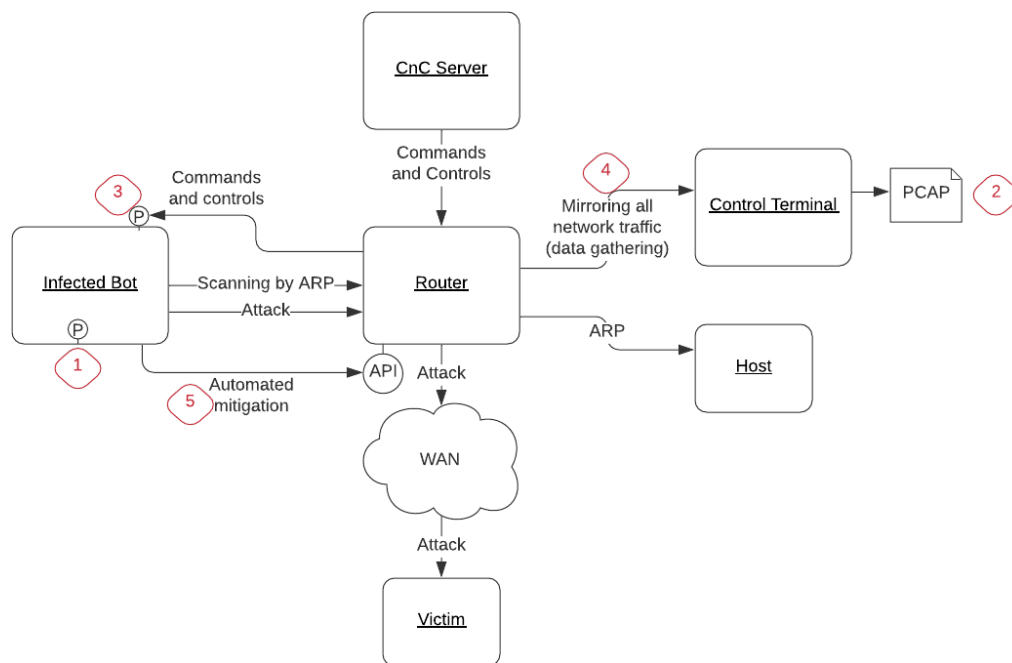


Figure 4.2: Implementation approach for the different parts

1. Host disinfection process. Even though Mirai takes various precautions not to be detected, there are certain procedures one can follow. Following these, the search can be narrowed down to a few processes. Subsequently, if confirmed, the harmful processes can be stopped and the device can be temporarily disinfected.

2. Retrospective analysis. In the retrospective analysis the collected network data is analyzed. The goal is to be able to determine, after defining certain thresholds, whether there has been Mirai activity and, if so, to what extent.

3. Real-time analysis. In order to react more timely to attacks, other approaches must be sought. Therefore, a part of the implementation includes real-time analysis and detection. The method for the real-time analysis benefits from the insights gained in the retrospective analysis and leverages similar detection methods.

4. Data gathering. As a basis for many tasks, but particularly for the retrospective analysis, all network packets must be monitored and captured.

5. Automated mitigation. The detection of an attack does not yet prevent it. In order to be able to take remedial actions with the lowest possible latency, certain tools are leveraged, such as the router's Application Programming Interface (API).

Chapter 5

Implementation

This Chapter shows the technical implementation of this work. The Sections of this Chapter outline the adaption to the original Mirai code needed, the implementation of the MySQL database, the CnC server and the bot code. At the end, the implementation of the traffic analysis is presented.

5.1 Mirai

Before being able to make the published source code of Mirai run, some adaptations have to be made. Some of the bugs in the code released on GitHub [9] were already documented as by Sinanovic and Mrdovic (2017) [29] and others appeared during the implementation process. In order to conduct the analysis as planned, the following bugs had to be fixed.

5.1.1 Bug Fixes

- **Locked table.** Mirai's code has many tables where it writes and retrieves information. Such does also the bot module which is the part of Mirai running on the infected devices. One particular table is the `TABLE_KILLER_STATUS` storing information about the *killer* process. The killer process is responsible for killing processes which use the ports needed for communicating with the CnC server [29].

This also includes killing other instances of Mirai, if detected. This particular table is accessed in the killer module *killer.c* in line 182 without first unlocking the table which results in an error. The bug could be fixed by first unlocking and finally locking the table again as illustrated in listing 5.1. The two steps are applied in line 2 and line 5 of listing 5.1.

```

1 //In the following line the table is unlocked
2 table_unlock_val(TABLE_KILLER_STATUS);
3 ptr_status_path += util_strcpy(ptr_status_path, table_retrieve_val(
    TABLE_KILLER_STATUS, NULL));
4 //In the following line the table is locked again
5 table_lock_val(TABLE_KILLER_STATUS);

```

Listing 5.1: Lock and unlock the table TABLE_KILLER_STATUS

- **Attack prevention in debug.** Per default, the bot module of Mirai can be built for debug or release. While in debug there are many messages logged onto the terminal, the release should provide production-ready, stripped binaries which then can be loaded onto newly infected devices. When compiled for debug, there are several mechanisms within the code to *prevent* starting attacks. This forms a security feature from the attacker's perspective and should help not to get detected. In order to profit from the feedback in form of log messages, the attack prevention mechanisms in debug mode were disabled by commenting out the following lines of code shown in listing 5.2.

```

1 #ifdef DEBUG
2 break;
3 if (errno != 0)
4 printf("errno = %d\n", errno);

```

Listing 5.2: Disabling the attack prevention in debug

5.1.2 Database

As previously mentioned, the operation of Mirai requires a MySQL database to be in place [29]. Next to administrative tables like *users* or *attack history*, the database saves operational information as for example the domain for CnC or the reporting server. A total of 51 such operational tables are defined in the bot module of Mirai [9]. To enable the CnC server to log into the database and perform operations, the database information and credentials need to be edited in the CnC's main file *main.go*. An example of how this could look like is given in listing 5.3.

```

1 const DatabaseAddr string = "127.0.0.1:3306"
2 const DatabaseUser string = "dbuser"
3 const DatabasePass string = "dbpassword"
4 const DatabaseTable string = "mirai"

```

Listing 5.3: Credentials and information of the database 'mirai'

After the database is set up, the three persistent tables need to be created. Next to the *history* table shown in Table 4.1, one table for the *users* and another for a *whitelist* are needed. In listing 5.4 we can observe a list of the first 5 attacks launched and persisted. With that command, a maximum number of bots can be specified. This is a value which is an attribute of the user and not, as one might expect, an attack parameter. Users also have attributes called *last_paid* and *duration_limit*. It is to be assumed that a botnet leasing mechanism was originally intended. They could specify a maximum number of bots available for a user which depends on the amount being paid.

```
1 mysql> select * from history where id < 6;
2 +-----+-----+-----+-----+-----+-----+-----+
3 | id | user_id | time_sent | duration | command | max_bots |
4 +-----+-----+-----+-----+-----+-----+-----+
5 | 1 | 1 | 1570720215 | 10 | udp 192.168.10.7 10 | -1 |
6 | 2 | 1 | 1570721105 | 5 | udp 192.168.10.7 5 | -1 |
7 | 3 | 1 | 1570791665 | 5 | udp 192.168.10.6 5 | -1 |
8 | 4 | 1 | 1570791701 | 5 | syn 192.168.10.6 5 | -1 |
9 | 5 | 1 | 1570791727 | 5 | syn 192.168.10.6 5 | -1 |
10 +-----+-----+-----+-----+-----+-----+-----+
11 5 rows in set (0.01 sec)
```

Listing 5.4: Attack history of the first 5 attacks

For the scope of this thesis, a user with credentials 'admin' / 'admin' was added to the users database. The account has editing rights and no upper limit in terms of number of bots. The insert statement is as follows:

```
1 INSERT INTO users VALUES (NULL, 'admin', 'admin', 0, 0, 0, 0, -1, 1, 30, '');
```

Listing 5.5: Inserting statement for the user table

5.1.3 CnC Server

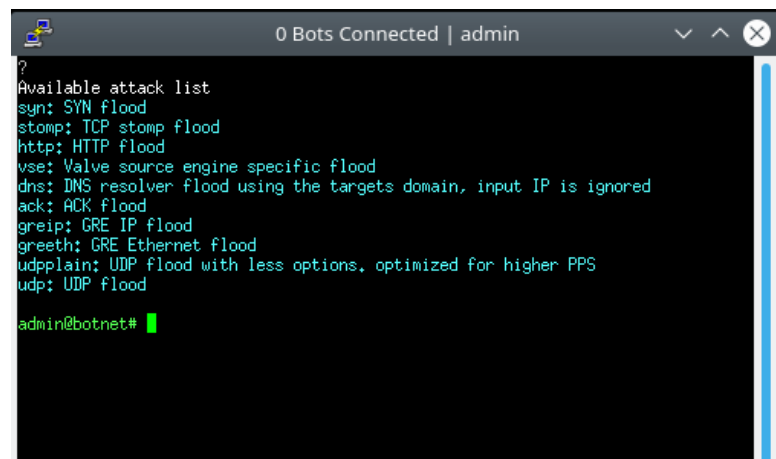


Figure 5.1: CnC server terminal after successful login showing all possible attack vectors

After applying the adaptations described in Section 5.1.1, the CnC server does not need any further changes or modifications. Executing the command `./build.sh release telnet` in the directory `/Mirai-Source-Code/mirai/`, a `/release` directory is created containing an executable file called `CnC`. When executed, the machine turns into the CnC server and listens to port 23 for any telnet connection of registered users. This process is depicted in Figure 5.1. By entering the help command `'?'` into the terminal, a list of attack vectors is presented. Additionally, the terminal toolbar of the CnC server gives continuous overview of the number of connected bots, which can be retrieved from Figure 5.1.

5.1.4 Custom Encoding

There are various built-in obfuscation mechanisms in Mirai which should prevent detection effectively. One prominent feature is the custom encoding method which gets compiled along with the CnC server and the bot module. It belongs to the set of internal instruments of Mirai and is located in the `/tools` directory. The syntax of using the `ENC` executable is as follows:

```
./enc <string | ip | uint32 | uint16 | uint8 | bool> <data to obfuscate>
```

For example, encoding the fictive domain `'myreporting.com'` results in the command `./enc string myreporting.com` with output:

```
\x4F\x5B\x50\x47\x52\x4D\x50\x56\x4B\x4C\x45\x0C\x41\x4D\x4F\x22
```

It is worth mentioning that the encoder is only compiled when building for debug. The reason behind this is that any string to encode needs to be encoded *before* the actual attack or the spreading of Mirai is started. Thus, when launched for release, the bots already need to know the domain of the CnC and reporting server.

5.1.5 Bots

The bot module is the most versatile and most complex among the Mirai code modules. The area of responsibility of the bot module covers scanning for new devices, trying to brute force new devices, report them to the reporting server, connect to the CnC server and listen for commands and finally, if ordered, launch attacks against victims.

In order to complete all tasks as intended, a certain degree of adaption and modification is required. Especially when it comes to communication with other parts of Mirai, as the CnC server or the reporting server. The bot lifecycle begins when it initially starts scanning. There measures which prevent scanning certain IP addresses. Initially, this mechanism was designed to prevent getting detected by scanning U.S. federal institutions. Examples for that are the Federal Postal Service or General Electrics (GE). For the purpose of this thesis, the allowed subnet to be scanned was limited to 192.168.0.0/16, while all devices were either in 192.168.10.0/24 or 192.168.20.0/24.

By continuously sending Address Resolution Protocol (ARP) requests, bots scan through the allowed range of allowed IP addresses. Any answer to such a request is interpreted as an active host which communicates its Media Access Control (MAC) address. The bot then tries to connect to the default telnet port 23, trying to establish a telnet session. Every 10th attempt is connecting to port 2323. If Telnet is not configured on the device, the port is most likely closed and the attempt will be discontinued. Otherwise, the active host will respond with the Telnet prompt after successful connection. The bot reads the prompt text stepwise and verifies based on the text on which stage of the login process it currently is. It successively sends combinations of a fixed set of credentials to the possible victim until either all possibilities are exhausted or one credential pair leads to a successful login. Due to faulty prompt strings read by the bot, this step of the autonomous propagation could not be recreated. To accomplish the purpose of this thesis, the hosts in the testbed were infected manually with a compiled version of the bot module.

The original set of credential pairs of Mirai contained 46 pairs [3], while the Mirai version published on Github [9] now contains 62 credential pairs. The evolution of this set is outlined in-depth by Antonakakis, Manos, et al. 2017 [3]. For the scope of this thesis, the set of credentials was altered to a total set of 4 username-password pairs. The insert commands of the complete set of credential pairs is depicted in listing 5.6.

```

1 add_auth_entry("\x4D\x56\x4A\x47\x50\x13\x22", "\x4D\x56\x4A\x47\x50\x13
  \x22", 1);          // other1   other1
2 add_auth_entry("\x4D\x56\x4A\x47\x50\x10\x22", "\x4D\x56\x4A\x47\x50\x10
  \x22", 1);          // other2   other2
3 add_auth_entry("\x4E\x4B\x4C\x43\x50\x4D\x22", "\x4E\x4B\x4C\x43\x50\x4D
  \x22", 1);          // linaro    linaro
4 add_auth_entry("\x4F\x4B\x50\x43\x4B\x40\x4D\x56\x57\x51\x47\x50\x22", "
  \x4F\x4B\x50\x43\x4B\x22", 1);          // miraibotuser  mirai

```

Listing 5.6: The insert commands of the complete set of credentials used for this thesis

The credential set consists of two dummy credentials (lines 1-2), and two existing account credentials. This holds for all targets in the testbed. It is to be mentioned that, like any other information stored or retrieved within Mirai, the credentials have to be encoded before altering the insert commands.

Mirai Module	Domain	IP Address
CnC Server	mycnc.com	192.168.10.254
Reporting Server	myreporting.com	192.168.10.254

Table 5.1: The static IP table used by the DNS server

After successful login, the bot follows a defined victim reporting procedure. The reporting server is separated from the CnC server and also has a different domain name entry in the Mirai code. Therefore, it most likely ran under a different IP address when Mirai was used in the past. The static IP table used by the DNS in this thesis is shown in Table 5.1. Since both servers are running on the same machine, both domains are resolved to the same IP address. The reporting server is configured to listen to port 48101 as default and waits for the bots to report vulnerable devices in the defined syntax of `<IP address> <username>:<password>`. After successful report, the CnC server launches to loader to infect the device.

5.2 DNS Server

Since the experimental setup is operating in a local network without any internet connection, every DNS request needs to be resolved differently. Therefore, the DNS feature of the router's firmware was used. Any domain used in the Mirai code was matched to a static IP address which the domain was resolved to. In order to send all DNS requests to the router only, a Network Address Resolution (NAT) rule had to be created. By redirecting all UDP packets with destination port 53, which is the default DNS port [23], the router receives all outgoing DNS requests.

5.3 Host disinfection

Similar to other malware, Mirai hides from detection systems. This is not only limited to the network but includes also host-based intrusion detection. One prominent example for this is Netstat [24] which is a common tool for displaying the contents of network-related data. Mirai has a built-in mechanism to hide parts of its bot processes from Netstat. This means that bits of Mirai which run on an infected device are being obfuscated when using Netstat. Precisely, Mirai's bot module is divided into two parts. One part is responsible for Mirai's perpetual propagation by scanning and bruteforcing hosts in the network. The other part is steadily listening on a dedicated port 101 for API calls of the CnC server in

order to start an attack. While the first part, occupying the default telnet port 23, is listed by Netstat, the latter is not. The hiding is also indicated when a user is connecting to the CnC server, which in turn has connections to all bots. In the connection establishment process the console reads "Hiding from netstat...", which it does successfully. In order to achieve a comprehensive host clean up, Netstat does not suffice. One helpful fact about Mirai is that the same process name is used for both parts of the bot module. It uses a 14 character long process name which is a random composition of letters and numbers. This, in combination with other indicators like the high CPU usage, helps to nail down the Process Identifier (PID). Listing 5.7 shows how the two parts, i.e. processes, can be identified, killed and the victim therefore be rescued from the infection.

```

1 # write all (PName / PID) pairs related to the IP into the file
2 os.system("sudo netstat -tnp | grep '" + packet.ip.dst + "' | awk '{
   print $7}' > processes.txt")
3 f = open("processes.txt","r")
4 f1 = f.readlines()
5 for process in f1:
6     pid = process.split("/")[0]
7     pname = process.split("/")[1][: -1]
8     if int(pid) != 1:
9         # kill the process which is scanning
10        os.system("sudo kill " + pid)
11
12        # find the process which attacks the victim
13        os.system("sudo top -n 1 | head -10 | tail -3 | awk '{print $10,
   $2, $13}' > top3processes.txt")
14        top3proc = open("top3processes.txt", "r").readlines()
15        for proc in top3proc:
16            cpuload = proc.split(" ")[0]
17            procname = proc.split(" ")[2]
18            # process names are always 14 letters long
19            if float(cpuload) > 90 and len(procname) > 13:
20                # kill process which is attacking
21                os.system("sudo kill " + proc.split(" ")[1])
22                break

```

Listing 5.7: How to clean up a host from Mirai processes

5.4 Retrospective Analysis (PCAP Analysis)

There are various indicators which can be subject to a Mirai detection mechanism. Although Mirai is designed to be as obfuscated as possible, it causes suspicious activity which stays detectable. This includes not only a flood of ARP packets caused by Mirai's scanning mechanism, but also resolved DNS requests, started Telnet sessions and CnC attack commands. While some are clear indicators for Mirai activity, others may just hint to doubtful traffic within the network.

The Mirai retrospective analysis script is another outcome of this thesis. It takes a packet capture file (.pcap) as input and aggregates the before mentioned hints. The console output gives a holistic overview of any Mirai-related data. The key figures which are analyzed are composed as follows: If a host in the network exceeds a certain threshold for ARP packets, it is marked as scanning. The threshold is a combination of an absolute number and the average number of ARP packets per host having sent at least one ARP packet. The absolute number is defined to be 500, as this amount of ARP packets is already exceeded after a few seconds of scanning. As illustrated in listing 5.8, the array `arp_cand` contains all IP addresses which could be considered as scanning.

```

1 for packet in packetsPyshark:
2     if packet.highest_layer == "ARP":
3         arpcounter_total += 1
4         addToDict(arp_overview, packet.arp.src_proto_ipv4)
5 arp_threshold = arpcounter_total / len(arp_overview)
6 arp_cand = [arpsender for arpsender in arp_overview if arp_overview[
    arpsender] > arp_threshold && arp_overview[arpsender] > 500]

```

Listing 5.8: Collecting and analysing ARP packets

Furthermore, all Telnet sessions are reconstructed by the sequence number. Since every infected bot has to connect to a CnC server at least once, every IP address which started a Telnet session is saved, listed separately and cross-checked with the preceding list of allegedly scanning IP addresses. Since the Telnet communication between bots and other parts of the botnet is essential, all the data is written into a SQLite database. This allows for more efficient evaluation and querying of the data, if needed. Listing 5.9 shows how packets are identified as Telnet communication, stored into the database and queried afterwards. The very first packet of a cohesive communication stream is identified by grouping all packets by the sequence number and then selecting the smallest packet frame number. This approach is shown in line 11 of listing 5.9.

```

1 for packet in packetsPyshark:
2     # save all telnet packets
3     if hasattr(packet, "tcp") and hasattr(packet, "telnet") and (packet.
        tcp.dstport == "23" or packet.tcp.srcport == "23"):
4         for layer in packet.layers[::-1]:
5             if layer.layer_name == "ip" :
6                 write_TelnetPack(conn, (packet.tcp.stream, packet.number, layer.src
                    , layer.dst, packet.telnet.data))
7                 break
8 # get all IP addresses which started a Telnet session
9 cur = conn.cursor()
10 cur.execute("select DISTINCT src from Telnet_packs group by seqNr having
    min(frameNr);")
11 return cur.fetchall()

```

Listing 5.9: Reconstructing Telnet sessions and extracting initiating IP addresses

```
Most suspicious IPs are:
[+] 192.168.10.4
    which fetched following domains:
    [-] mycnc.com
    [-] 3.10.168.192.in-addr.arpa
    [-] 4.10.168.192.in-addr.arpa
    [-] 8.10.168.192.in-addr.arpa
[+] 192.168.10.3
    which fetched following domains:
    [-] mycnc.com
    [-] 3.10.168.192.in-addr.arpa
    [-] 4.10.168.192.in-addr.arpa
[+] 192.168.10.8
    which fetched following domains:
    [-] mycnc.com

Most suspicious Domains are:
[+] mycnc.com
```

Figure 5.2: Extract from the PCAP analysis script

At this point, the intersection of the two indicators is considered. This includes every IP address, having allegedly been scanning and having started a Telnet communication. For this subset, all former DNS requests are searched and listed. The domain which needs to be contacted by the bot is shipped with the malicious Mirai code in a custom encoded format. By listing all domains by the allegedly infected hosts, doubtful domains can be identified. As a last step of this part, the domains are compared and the intersection is displayed as 'the most suspicious domains'.

Last but not least, all Mirai-specific attack commands are parsed, analyzed and displayed. This is possible because of two facts. First of all, the source code has been published and is therefore available for any reverse engineering work. Secondly, the Telnet variant of Mirai has been used throughout this thesis. As the Telnet protocol transmits data unencrypted, it is possible to parse commands and react upon payload information. The Telnet variant of Mirai was used in 26.44% of the cases making it the second most used variant of Mirai ever and the single most variant of the initial release of Mirai [3].

```
#### Mirai Attack Overview ####
IP address '192.168.10.7' attacked '192.168.10.4' for 2s with a 'ACK flood attack'
IP address '192.168.10.7' attacked '192.168.10.4' for 2s with a 'ACK flood attack'
IP address '192.168.10.7' attacked '192.168.10.4' for 2s with a 'UDP Plain flood attack'
IP address '192.168.10.7' attacked '192.168.10.4' for 2s with a 'UDP Plain flood attack'
IP address '192.168.10.7' attacked '192.168.10.4' for 2s with a 'UDP flood attack'
IP address '192.168.10.7' attacked '192.168.10.4' for 2s with a 'UDP flood attack'
IP address '192.168.10.7' attacked '192.168.10.4' for 2s with a 'SYN flood attack'
IP address '192.168.10.7' attacked '192.168.10.4' for 2s with a 'SYN flood attack'
```

Figure 5.3: Aggregating all Mirai attack commands found in the PCAP file

The script displays the IP address of the CnC server, i.e. the source of the command, the command recipient (bot), the attack duration and the type of attack. Apart from standard packet frame information, the before mentioned parameters form the main body in the payload of a command. The IP suffix of the attack victim is also part of the payload. If the suffix is other than 32, a subnet is target of the attack rather than a single IP address. The attack command is visible in a hexadecimal format and can be parsed, as it is shown in Figure 5.4.

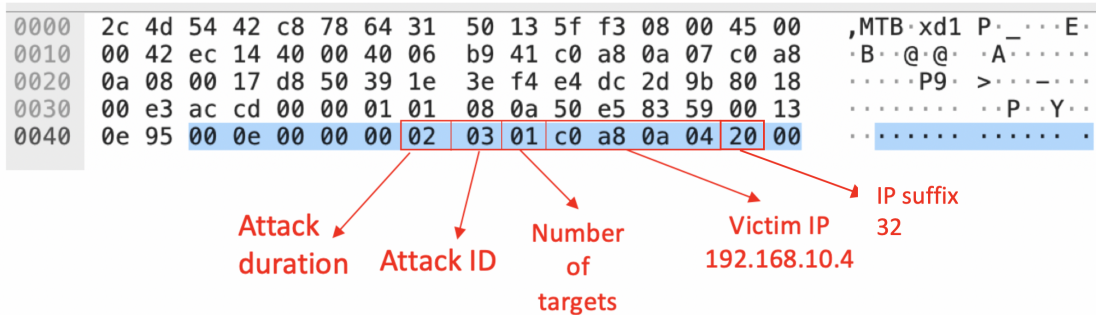


Figure 5.4: Parsing an attack command in a hexadecimal form

5.5 Real-time analysis

In order to analyze real-time traffic with minimal latency, a similar approach is taken. The syntax of Mirai commands is as shown in Figure 5.4. If some of the information can be detected, it is certain that the packet is related to a Mirai attack command. The un-encrypted format of the Telnet payload is essential for real-time analysis. The lower the operational effort to parse verifying information, the lower the latency and therefore a shorter time needed to take necessary countermeasures.

The code snippet shown in listing 5.10 illustrates how Mirai characteristics can be leveraged for its detection. Such as the specific syntax used to launch commands, every packet

carrying attack information has the exact length of 80. After the first check, the victim's IP address is searched within the payload. By using Python's socket library, the data is verified to be an IP address. In case packets carry other payloads, the operation will fail triggering a handled exception. This approach causes minimal latency and allows for fast Mirai command detection.

```

1 import socket
2
3 def check_telnet(packet):
4     if packet.highest_layer == "TELNET" and int(packet.captured_length)
       == 80:
5         addr_hex = packet.tcp.payload.split(":")
6         addr_dec = [str(int(h,16)) for h in addr_hex]
7         addr = ".".join(addr_dec[8:12])
8         try:
9             socket.inet_aton(addr)
10            print("# Addr in payload detected! Destination: " + packet.
ip.dst)
11        except:
12            print("Telnet packet not a command.")

```

Listing 5.10: Function to run real-time analysis for Mirai attack detection

The function shown in listing 5.10 uses the Pyshark [25] library to detect attacks and react after the attack is verified. The code snippet shown in 5.11 imports and calls the function while applying it to every incoming packet at the same time. The packet capturing starts on the Ethernet interface `eth0`. In this example, no action is taken upon detection of a command. There is only a message printed to the console. Some of the mitigating measures are explained and illustrated in Section 5.6.

```

1 from live_functions import check_telnet
2 import pyshark
3 capture = pyshark.LiveCapture(interface='eth0')
4 capture.apply_on_packets(check_telnet)

```

Listing 5.11: Pyshark code used to start the real-time analysis

5.6 Automated implementation of mitigating measures

The mitigating measures in this thesis are actions taken after successful detection of Mirai activity or after identification of doubtful traffic in the network. Depending of the level of certainty in the verification process, different steps can be taken. In case of a doubtful behavior of an allegedly infected host in the network, the amount of outgoing data could be

limited by a queue. This would ensure that possible attacks from this host are prevented or at least delimited in a timely manner.

APIs provide fast access to the routing logic. There are many possibilities to create, change or remove configurations which have influence on how the network behaves. The data flow can be adapted programmatically as required. This can be particularly helpful when anomalies are detected and initial measures must be taken before further steps can be implemented.

5.6.1 Queues

Queueing refers to the shaping of the network flow according to given conditions. Queues have the ability to restrict individual participants in the network. The data flow can either be restricted absolutely, i.e. according to the number of packets or the number of bytes transferred. Or, the data flow is restricted by a relative size like the number of bytes transferred per second.

Mikrotik's API allows users to interact with RouterOS, change configurations and manage the router [20]. In order to use the RouterOS API with Python, Socialwifi's RouterOS API library [27] was used. As shown in listing 5.12, after successful authentication, an API object can be fetched and used for giving commands to the router. In lines 5 and 6, it is shown how a list of the existing queues is fetched and extended by a new one. The new queue targets the recipient of the currently analyzed packet. It is assumed that the recipient is an infected bot which, after receiving the packet, will start an attack against the specified victim. The queue imposes a limit of 10 Mbit/s on the target.

```

1 import routeros_api
2
3 connection = routeros_api.RouterOsApiPool('192.168.10.1', username='
    admin', password='', use_ssl=False)
4 api = connection.get_api()
5 list = api.get_resource('/queue/simple')
6 list.add(target=packet.ip.dst,total_burst_threshold="10M",
    total_max_limit="10M" )
7 print("Queue created successfully!. Target: ", packet.ip.dst)

```

Listing 5.12: Connecting to the router via RouterOS API

5.6.2 Firewall filters

Firewall filters are the most efficient measure when an intrusion has been confirmed. Existing connections can be disconnected and illegal data flows are prevented as soon as the filter comes into effect.

Similar to the creation of queues described in Section 5.6.1, after successful authentication, an API object can be fetched and used for giving commands to the router. In the array named `list` in line number 5 of listing 5.13, all current firewall filters are fetched and stored. In the last line, the current set of firewall filters is extended by a new one. For every packet on the forward chain coming from the specified IP address, the packet should be dropped and a log message should be created with the according prefix. The target of this filter is the recipient of the attack command coming from the CnC server. The recipient is assumed to be an infected bot which is about to start an attack and needs to be isolated. This goal is achieved with the filter specified in listing 5.13.

```
1 import routers_api
2
3 connection = routers_api.RouterOsApiPool('192.168.10.1', username='
    admin', password='', use_ssl=False)
4 api = connection.get_api()
5 list = api.get_resource('/ip/firewall/filter')
6 list.add(action="drop", src_address=packet.ip.dst, chain="forward", log=
    "yes", log_prefix="[CnC firewall rule]")
7 print("Filter created successfully!. Target: ", packet.ip.dst)
```

Listing 5.13: Creating remote firewall filters via RouterOS API

Chapter 6

Evaluation

This Chapter aims at the evaluation of efficiency and effectiveness of the implementation presented in Chapter 5. It should deliver confidence in how reliable and in which level of latency the proposed solutions work with. Reliability is derived from the number of successful implementation of the mitigation measures. This helps assessing whether similar techniques and approaches are suggestive for early botnet detection. In Section 6.2, the retrospective analysis script presented in Section 5.4 is evaluated. For that, test cases have been defined and conducted. The real-time packet analysis and attack command detection from Section 5.5 is then assessed in Section 6.3. The subsequent mitigation actions specified in Section 5.6, are then evaluated for efficiency and effectiveness in Section 6.4.

6.1 Evaluation Setup

The evaluation set-up follows the same architecture as the testbed scenario, described in Section 4.2 and depicted in Figure 4.1. It also includes the same devices with unchanged configuration. The scripts shown in Chapter 5 were coded in Python programming language version 3.6.9 [26]. The router, which was running MikroTik's RouterOS version 6.42.4 [22], was remotely called using the RouterOS API library version 0.17.0 by Social-wifi [27]. The network traffic has been captured using Wireshark version 2.6.10 [33].

The test configuration includes a single router. Because of the co-existence of two networks, the router is assigned with two IP addresses in these networks. This prevents switching on the MAC address (layer 2 switching) and ensures firewall filters and queues to be applied accordingly. The control terminal sends the attack command to one or two bots as required. These bots then start the corresponding attack. The victim of the attack is at least one host in the other network.

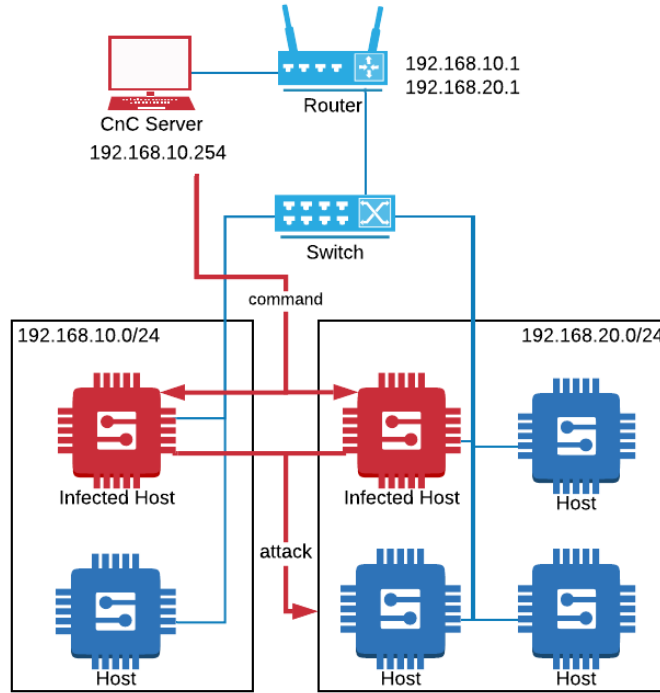


Figure 6.1: Overview of the evaluation setup including all involved components

6.2 Retrospective Detection

To evaluate the retrospective Mirai analysis in detail, a series of scenarios were defined. The main subject of the evaluation scenarios is on the functionality presented in Section 5.4. The traffic is captured using Wireshark and analyzed afterwards.

6.2.1 Key Performance Indicators and Thresholds

For the retrospective analysis, four main Key Performance Indicators (KPIs) were defined. Each of the indicators must meet a different threshold. The KPIs and their corresponding thresholds are explained individually as follows.

Detection of scanning activity. By counting the amount of ARP packets in a PCAP file, scanning activity can be identified. In the analysis script of this thesis, as mentioned in Section 5.4, the threshold consists of a relative and an absolute part. If the amount of ARP packets sent by a single IP address surpasses an absolute number of 500 and concurrently surpasses the ratio of the total amount of ARP packets divided by the number of hosts having sent at least one ARP packet, the IP address is marked as having been scanning and therefore *detected*.

Detection CnC communication. The detection of communication between infected bots and at least one CnC server is based on the fact that each bot has to keep an intact connection with the bot. This connection has to be established once. As a first step in the reconstruction, all Telnet packets are stored in a database. There are two main factors relevant for their classification:

- **Sequence Number:** The sequence number determines how the packets belong to each other. In particular, packets can be assigned to the communication they belong to.
- **Frame Number:** By means of the frame number, the packets can be ordered chronologically. An IP address is classified as detected by the following threshold. If a packet has the lowest frame number within a group of Telnet packets with the same sequence number, we assume that the sender of this packet has started the communication. The sender IP is stored and marked as *detected*.

Detection of the CnC's domain. The possible identification of the CnC IP address is another KPI and should, if possible, help finding out the origin and control point of the bots. Since the IP addresses behind the domain can change constantly, it is advantageous to block the domain rather than the IP address. If an IP address matches the two previous indicators, all DNS requests for this IP are searched and listed. The intersection of all these domains is also displayed separately as the 'most suspicious domains' (*cf.* Figure 5.2). It is to verify manually whether this list of domains then contains the domain of a CnC server.

Detection of attack commands. The detection of attack commands is the most meaningful KPI. According to Figure 5.4, a packet is marked as 'detected' as soon as an IP address is identified at a certain position within the payload. This syntax is considered to be specific for the Mirai variation used in this thesis [9], which hardly gives false positive results.

6.2.2 Scenarios

Single bot, same attack vector. The first of two test scenarios for the retrospective analysis consists of a single bot that uses the same attack vector multiple times. The User Datagram Protocol (UDP) flood, which is the attack vector with the highest reliability and attack bandwidth, was chosen for that scenario. The results are presented in Table 6.2.2. With few exceptions, most indications for Mirai activity have been identified. On average 80% of the patterns were detected. The detection of the CnC domain was successful for 60% of the cases. The reason is the dependence between scanning and CnC communication. Only the IP addresses being detected in both, in the scanning and in the CnC communication, are examined.

Attack Vector	Scanning	CnC Communication	CnC Domain	Attack
UDP (1)	Detected	-	-	Detected
UDP (2)	Detected	Detected	Detected	Detected
UDP (3)	Detected	Detected	Detected	Detected
UDP (4)	-	Detected	-	Detected
UDP (5)	Detected	Detected	Detected	Detected

Table 6.1: Results of the retrospective analysis evaluation using a single bot

Two bots, different attack vectors. The second test scenario includes two bots within two different subnets. The bots are ordered to use each of the 10 attack vectors once. The detection of attack commands again proved to be the most reliable detection. The results of this scenario are depicted in Table 6.2.2. In each of the 10 attacks, the command was intercepted and stored. However, the detection of scanning devices is difficult when more than one device is involved. In all executed tests, only the IP address that started scanning earlier was detected. This indicates a weakness in the detection of bots that are scanning. The identification of CnC communication was 80%. The dependence of the CnC domain identification on the previous results explains the 70% result there.

Attack Vector	Scanning	CnC Communication	CnC Domain	Attack
ACK	Partly	Partly	-	Detected
DNS	Partly	Detected	-	Detected
GRE ETH	Partly	Detected	Detected	Detected
GRE IP	Partly	Detected	Detected	Detected
HTTP	Partly	Detected	Detected	Detected
STOMP	Partly	Detected	Detected	Detected
SYN	Partly	Partly	-	Detected
UDP	Partly	Detected	Detected	Detected
UDP PLAIN	Partly	Detected	Detected	Detected
VSE	Partly	Detected	Detected	Detected

Table 6.2: Results of the retrospective analysis evaluation using two bots

6.3 Real-time detection

Real-time detection is the most important type of detection in terms of bringing the most value in practice. In order to evaluate the real-time detection script presented in Section

5.5, the following scenarios were chosen. The first five attack vectors, as shown in Figure 5.1, whereas each attack was launched two times and was examined for detection. In order to improve the verification of an attack, the code presented in Section 5.5 was modified as follows. Whenever a packet is classified as *detected*, a file is created. The file contains a timestamp and the packet which was captured and classified as an attack.

```

1 import os
2 import socket
3 from datetime import datetime
4
5 def check_telnet(packet):
6     if packet.highest_layer == "TELNET" and int(packet.captured_length)
       == 80:
7         starttime = datetime.now()
8         print("Telnet packet length: %s, tcp payload: %s" % (packet.
captured_length, packet.tcp.payload))
9         addr_hex = packet.tcp.payload.split(":")
10        addr_dec = [str(int(h,16)) for h in addr_hex]
11        addr = ".".join(addr_dec[8:12])
12
13        try:
14            socket.inet_aton(addr)
15            endtime = datetime.now()
16            myfile = open("RT-detection.txt", "a")
17            myfile.write("Time of detection: " + str(datetime.now()) + "
\n\n")
18            myfile.write("Packet captured: \n")
19            myfile.write(str(packet))
20            myfile.write("\n\n")
21            myfile.write("TCP Payload: ", str(packet.tcp.payload))
22            myfile.close()
23
24        except:
25            print("Something went wrong.")

```

Listing 6.1: Adaption of the real-time detection script for evaluation

In line 6 independent, sequentially executed tests, the attack command was detected in every case. This again confirms the reliability of the detection of attack commands, which was already apparent in Section 6.2. The results of the detection and the latency caused by the packet inspection are depicted in Table 6.3. The latency is calculated from the time the packet is being analyzed (line 5) to the confirmation that the packet is a Mirai attack command (line 12). The average latency was close to 1 millisecond.

Attack Vector	Detection	Latency (in ms)
ACK (1)	Detected	1.383
ACK (2)	Detected	1.553
DNS (1)	Detected	0.603
DNS (2)	Detected	0.887
GRE ETH (1)	Detected	0.568
GRE ETH (2)	Detected	1.041
HTTP (1)	Detected	1.238
HTTP (2)	Detected	0.501
SYN (1)	Detected	1.104
SYN (2)	Detected	0.927
UDP (1)	Detected	0.571
UDP (2)	Detected	1.599

Table 6.3: Results of the real-time detection evaluation

6.4 Mitigation Measures

The evaluation of queues and firewall filters as automated mitigation measures showed divergent results. At this point, it is important to point out that the results may be different with other routers. Namely, in this evaluation, the implementation of the firewall filter was only executed after the end of the attack. This could be explained by the fact that the router is overstrained with the attack and must wait for the end of the attack before it can execute commands. As a result, whenever an attack lasts longer than 5-10 seconds, the execution of a command to create a filter by the given API is aborted. To implement the filter successfully, this problem must be bypassed. This can be done either by repeated execution or by waiting for the attack to finish.

When implementing a queue, the results were different. The queue could be implemented repeatedly after an attack was detected and could have its effect already in the same attack. In Figure 6.2, the three attacks are visualized in two graphs. The upper graph is the byte graph and shows the number of Mbit/s (Mbps), while the packet graph below shows the absolute number of packets per second. The attacks are SYN flood attacks of 10 seconds each. The first attack shows the effect of an attack without any countermeasures. In the second attack, the real-time detection script runs with automatic implementation of a queue upon detection. The second spike shows that after a short latency time, the queue is successfully implemented and the entire data traffic is limited to 10 Mbit/s. This can also be seen in the code snippet in Listing 5.12. At the third attack, the queue is still active and restricts the traffic to the defined limit.

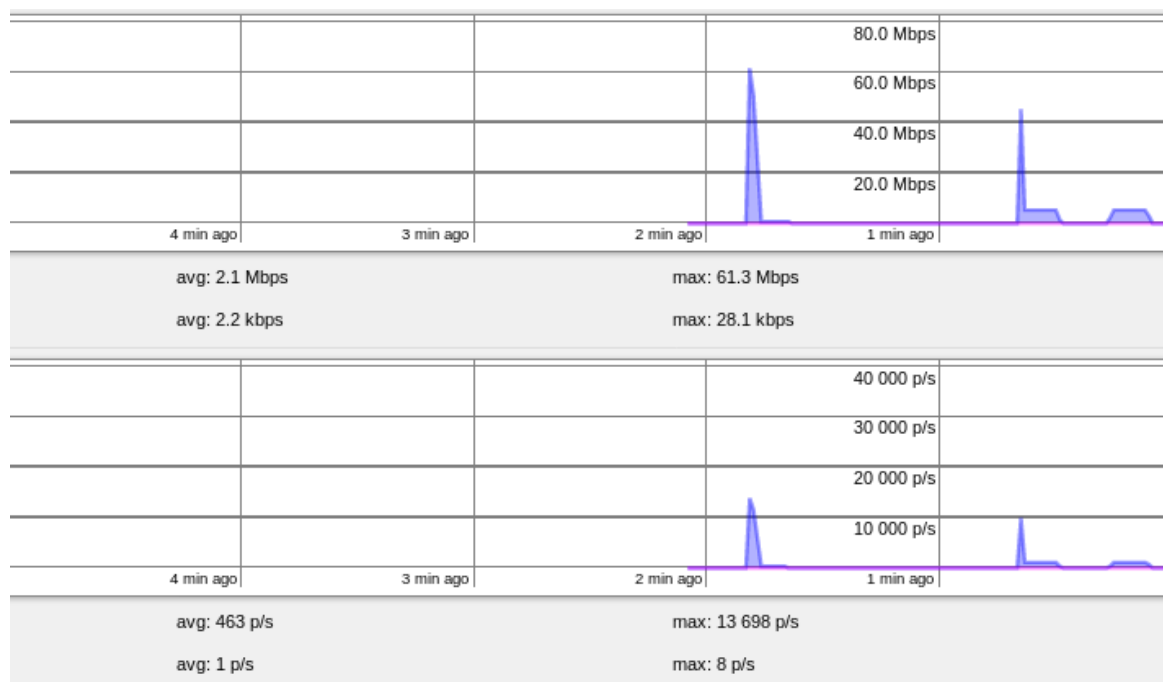


Figure 6.2: Evaluation of the automated queue implementation

Chapter 7

Conclusion

7.1 Summary and conclusion

The goal of this thesis was to analyse traffic related to the Command-and-Control (CnC) server to gain insight into the functioning of a botnet like Mirai. The insight would allow to develop detection techniques based on Mirai-specific characteristics. Reliable detection mechanisms can then be leveraged by security tools and functionalities such as firewalls which improves the overall security.

To achieve this goal, the source code of Mirai first had to be carefully analyzed. The resulting information is the basis for the detection of Mirai activity or Mirai attacks. The prototype for the retrospective analysis of Mirai traffic turned out to be effective. Besides the successful detection of all attack commands, 90% of the Telnet communications and half of the scanning devices were detected (Table 6.2.2). The basis for the subsequent real-time detection was the reliable detection of attack commands. This was used as a threshold to take action.

In the evaluation of the real-time detection of Mirai attack commands, each of the attacks was detected and correctly assigned. This includes the IP of the CnC server, the attack vector, the duration of the attack and the IP of the target. The router's API was leveraged to convert this knowledge into tangible measures. The automated implementation of firewall filters and queues showed the feasibility of defending Mirai. The effectiveness of these measures is evaluated and displayed in Section 6.4.

Specifically, queues and firewall filters have been evaluated as means for an automated mitigation measure upon detection of an attack. Nonetheless, the results were divergent because an automated mitigation is only partially feasible. While the implementation of queues is stable, firewall filtering usually requires waiting for the attack to end. Due to hardware limitations, the router is not able to process the API call until the attack is over. This is suboptimal for defending an attack from the beginning. This can be circumvented by performing the analysis on the router rather than mirroring all packets to the control terminal. This way, a filter can be created even before the attack starts.

Overall, the main goal of this thesis, namely the analysis of command and control data, was achieved. By analyzing Mirai program code, precise traffic patterns were determined according to its behavior. This allowed for evaluations after and during an attack. The implementation of defense measures was the continuation of this success, which only brings the actual benefit of defending and mitigating Mirai attacks.

7.2 Future work

The solutions proposed in this thesis take a signature-based detection approach. Since the source code of Mirai has been published, it is open and visible for everyone. Accordingly, the code can be reverse engineered and signatures, characteristic and syntax can be identified. For future work, a more general approach could be considered. By following an anomaly-based detection procedure, more general patterns could be learned and applied. This could be helpful in detecting other malware and variations of the Mirai source code. Also, the solutions of this thesis require lots of resources. Although the calculation effort, and thus, the latency is kept within reasonable limits, each and every packet is still analyzed individually. In order to develop meaningful applications for practical use, more resource-efficient solutions must be developed, which includes, for example, a method in which a randomised proportion of the packages, *e.g.*, by packet sampling, could be studied in future work.

Bibliography

- [1] UK National Crime Agency. *The Cyber Threat to UK Business*. 2016/17.
- [2] Esraa Alomari et al. “Botnet-based distributed denial of service (DDoS) attacks on web servers: classification and art”. In: *arXiv preprint arXiv:1208.0403* (2012).
- [3] Manos Antonakakis et al. “Understanding the mirai botnet”. In: *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 2017, pp. 1093–1110.
- [4] Paul Barford and Vinod Yegneswaran. “An inside look at botnets”. In: *Malware detection*. 2007, pp. 171–191.
- [5] Qian Chen and Robert A Bridges. “Automated behavioral analysis of malware: A case study of wannacry ransomware”. In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2017, pp. 454–460.
- [6] Christos Douligeris and Aikaterini Mitrokotsa. “DDoS attacks and defense mechanisms: classification and state-of-the-art”. In: *Computer Networks* 44.5 (2004), pp. 643–666.
- [7] *Ethereal/Wireshark*. URL: <https://wiki.mikrotik.com/wiki/Ethereal/Wireshark> (visited on Oct. 18, 2019).
- [8] Charles V Frank Jr. “Mirai Bot Scanner Summation Prototype”. In: (2019).
- [9] Jerry Gamblin. *Mirai-Source-Code*. 2017. URL: <https://github.com/jgamblin/Mirai-Source-Code>.
- [10] Jan Goebel and Thorsten Holz. “Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation.” In: *HotBots 7* (2007), pp. 8–8.
- [11] Guofei Gu et al. “Active botnet probing to identify obscure command and control channels”. In: *2009 Annual Computer Security Applications Conference*. IEEE. 2009, pp. 241–253.
- [12] Scott Hilton. *Dyn Analysis Summary Of Friday October 21 Attack*. 2016. URL: <https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/> (visited on Sept. 30, 2019).
- [13] *IDS vs. IPS: What’s the Difference?* June 28, 2019. URL: <https://www.dnsstuff.com/ids-vs-ips> (visited on Sept. 30, 2019).
- [14] Constantinos Kolias et al. “DDoS in the IoT: Mirai and other botnets”. In: *Computer* 50.7 (2017), pp. 80–84.
- [15] Ayush Kumar and Teng Joon Lim. “Early Detection of Mirai-Like IoT Bots in Large-Scale Networks through Sub-sampled Packet Traffic Analysis”. In: *Future of Information and Communication Conference*. Springer. 2019, pp. 847–867.
- [16] Hung-Jen Liao et al. “Intrusion detection system: A comprehensive review”. In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 16–24.

- [17] IP Location. *What is the difference between public and private IP address?* Nov. 18, 2018. URL: <https://www.iplocation.net/public-vs-private-ip-address> (visited on Oct. 3, 2019).
- [18] Knud Lasse Lueth. “State of the IoT 2018: Number of IoT devices now at 7B â Market accelerating”. In: *IoT Analytics* (Aug. 8, 2018).
- [19] Muhammad Mahmoud, Manjinder Nir, Ashraf Matrawy, et al. “A Survey on Botnet Architectures, Detection and Defences.” In: *IJ Network Security* 17.3 (2015), pp. 264–281.
- [20] MikroTik API Documentation. Nov. 8, 2019. URL: <https://wiki.mikrotik.com/wiki/Manual:API> (visited on Dec. 24, 2019).
- [21] MikroTik Packet Sniffer Feature. URL: https://wiki.mikrotik.com/wiki/Manual:Tools/Packet_Sniffer (visited on Oct. 18, 2019).
- [22] MikroTik RouterOS. URL: <https://mikrotik.com/software> (visited on Oct. 18, 2019).
- [23] Paul Mockapetris et al. “RFC-1035 Domain Names-Implementation and Specification”. In: *Network Working Group* 55 (1987).
- [24] Netstat. Apr. 17, 2019. URL: <http://man.openbsd.org/netstat.1> (visited on Dec. 22, 2019).
- [25] Pyshark. 2017. URL: <https://github.com/KimiNewt/pyshark/> (visited on Dec. 23, 2019).
- [26] Python - Programming Language. URL: <https://www.python.org/> (visited on Dec. 24, 2019).
- [27] Python RouterOS Library. Sept. 25, 2019. URL: <https://github.com/socialwifi/RouterOS-api> (visited on Dec. 24, 2019).
- [28] Luis Gustavo Araujo Rodriguez et al. “Analysis of vulnerability disclosure delays from the national vulnerability database”. In: *Anais do Workshop de Segurança Cibernética em Dispositivos Conectados (WSCDC-SBRC 2018)*. Vol. 1. SBC. 2018.
- [29] Hamdija Sinanovic and Sasa Mrdovic. “Analysis of Mirai malicious software”. In: *2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)* (2017), pp. 1–5.
- [30] Stephen M Specht and Ruby B Lee. “Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures.” In: *ISCA PDCS*. 2004, pp. 543–550.
- [31] *What is the Mirai Botnet?* URL: <https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/> (visited on Sept. 21, 2019).
- [32] Wikipedia contributors. *Botnet — Wikipedia, The Free Encyclopedia*. 2019. URL: <https://en.wikipedia.org/w/index.php?title=Botnet&oldid=916856149> (visited on Sept. 21, 2019).
- [33] Wireshark. URL: <https://www.wireshark.org> (visited on Oct. 18, 2019).

Abbreviations

API	Application Programmable Interface
ARP	Address Resolution Protocol
AV	Anti-virus system
CnC	Command-and-Control (Also C&C or C2)
DDoS	Distributed Denial-of-Service
DNS	Domain Name Server
DoS	Denial-of-Service
GB	Gigabyte
Gbit/s	Gigabits per second (Gpbs)
Ghz	Gigahertz
HIDS	Host Intrusion Detection System
HTTP	HyperText Transfer Protocol
ID	Identifier
IDPS	Intrusion Detection and Prevention System
IDS	Intrusion Detection System
IoT	Internet-of-Things
IP	Internet Protocol
IRC	Internet-Relay-Chat
ISP	Internet Service Provider
MAC	Media Access Control
Mbit/s	Megabits per second (Mbps)
NAT	Network Address Resolution
NIDS	Network Intrusion Detection System
OS	Operating System
OT	Operational Technology
P2P	Peer-to-Peer
PCAP	Packet CAPture
PID	Process Identifier
POP	Post-Office-Protocol
RAM	Random Access Memory
SSH	Secure Shell
Tbit/s	Terabits per second (Tpbs)
Telnet	Teletype Network

List of Figures

2.1	Botnet architecture including basic communication and operation [14] . . .	6
2.2	Basic DDoS architecture resulting from a botnet [19]	7
3.1	Trade-off between expensive sampling frequency and impactful detection delay for attack probabilities $p_1 = 0.5, 0.7, 0.9$ [15]	12
4.1	Overview of the experimental setup including all involved components . . .	17
4.2	Implementation approach for the different parts	19
5.1	CnC server terminal after successful login showing all possible attack vectors	23
5.2	Extract from the PCAP analysis script	29
5.3	Aggregating all Mirai attack commands found in the PCAP file	30
5.4	Parsing an attack command in a hexadecimal form	30
6.1	Overview of the evaluation setup including all involved components	36
6.2	Evaluation of the automated queue implementation	41

List of Tables

2.1	Comparison of the pros and cons of the signature-based and anomaly-based detection methods by Liao, Hung-Jen, et al. 2013 [16]	8
3.1	The most infected device vendors as a fraction of Mirai banners found by Censys [3]	9
3.2	The top Mirai victims in a seven-month observation period	10
3.3	Dataset summary from the <i>Bot Scanner Summation Prototype</i> for a selected date range	11
4.1	Mirai’s HISTORY table structure including fieldnames, types, keys and default values from Sinanovic and Mrdovic (2017) [29]	16
5.1	The static IP table used by the DNS server	26
6.1	Results of the retrospective analysis evaluation using a single bot	38
6.2	Results of the retrospective analysis evaluation using two bots	38
6.3	Results of the real-time detection evaluation	40

Listings

5.1	Lock and unlock the table TABLE_KILLER_STATUS	22
5.2	Disabling the attack prevention in debug	22
5.3	Credentials and information of the database 'mirai'	22
5.4	Attack history of the first 5 attacks	23
5.5	Inserting statement for the user table	23
5.6	The insert commands of the complete set of credentials used for this thesis	25
5.7	How to clean up a host from Mirai processes	27
5.8	Collecting and analysing ARP packets	28
5.9	Reconstructing Telnet sessions and extracting initiating IP addresses . . .	28
5.10	Function to run real-time analysis for Mirai attack detection	31
5.11	Pyshark code used to start the real-time analysis	31
5.12	Connecting to the router via RouterOS API	32
5.13	Creating remote firewall filters via RouterOS API	33
6.1	Adaption of the real-time detection script for evaluation	39

Appendix A

Usage guidelines

A.1 Retrospective analysis

The following libraries are required in order to execute the retrospective analysis:

- pyshark
- time
- socket
- os
- datetime
- sqlite3

The script file is called `Retrospective_analyzer.py`. When the data gathering is terminated and the data saved into a PCAP file, it has to be stored within the same directory as the analyzer file, namely within a directory called `/sources`. The file structure is as follows:

```
/
.. Retrospective_analyzer.py
.. /sources
... network-data.pcap
```

In line 10 of the `Retrospective_analyzer.py`, a string variable is set to the filename which is subject of the analysis. In order to scan the fictive file `network-data.pcap`, line 10 should look as follows:

```
1 fileName = "sources/network-data.pcap"
```

After that, the file can be executed. The output will be printed to the console.

A.2 Real-time Analysis

For the real-time analysis and remote configuration of a Mikrotik router, the following libraries are required:

- pyshark
- os
- time
- socket
- routeros_api
- datetime

The file called `Live_function.py` stores a method, which is the main function for the real-time detection. The method is called by the script `Live_analyzer.py`. There are 3 variations of the live analyzer contained:

Live_function_plain.py

The *plain* variation of the live function will print some specified log-type messages. This should help the user and be an orientation on how the analysis is proceeding. In case a Mirai attack command should be detected, a message would be printed to the console accordingly.

Live_function_filter.py

The *filter* variation of the live function is preconfigured to automatically implement a firewall filter on a Mikrotik router, as soon as a Mirai attack command is detected. The router's IP address and the connection parameters are specified in line 17 of the function file. In the test cases, the connection parameters looked as follows:

```
connection = routeros_api.RouterOsApiPool('192.168.10.1', username='admin', password='', use_ssl=False)
```

The new filter is added to the existing list of filters on the forward chain with the action *drop* and a specified log prefix. This can be changed in line 21 and looked as follows:

```
list.add(action="drop", src_address=packet.ip.dst, chain="forward", log="yes", log_prefix="[CnC firewall rule]")
```

By starting the script `Live_analyzer.py`, the real-time analysis is started.

Live_function_queue.py

The *queue* variation of the live function is preconfigured to automatically implement a queue on a Mikrotik router, as soon as a Mirai attack command is detected. The router's connection parameters are the same as in the *filter* variation, namely on line 17. The

new queue is configured to limit the bandwidth on line line 21 to an upper limit of 10 Mbit/s and looks as follows:

```
1 list.add(target=packet.ip.dst,total_burst_threshold="10M",  
    total_max_limit="10M" )
```

Live_analyzer.py

In the live analyzer, the interface the live analysis is be performed on, can be changed in line 7. In order to analyse all incoming packets on interface `eth0`, the line should look as follows:

```
1 capture = pyshark.LiveCapture(interface='eth0')
```


Appendix B

Contents of the CD

The following content is part of the enclosed CD.

Documentation

- Thesis as LaTeX source code including all figures
- Thesis as PDF
- Intermediate and final presentation (PPTX)

Implementation

- Mirai source code in directory /mirai
- Python source code for retrospective analysis
- Python source code for real-time detection
- Python source code for host cleanup

Evaluation

- Sources (PCAP files) and results (TXT files) of the evaluation of retrospective detection with 1 bot and 1 attack vector which is presented in section 6.2.
- Sources (PCAP files) and results (TXT files) of the evaluation of retrospective detection with 2 bots and 10 attack vectors which is presented in section 6.2.
- Results (TXT files) of the evaluation of real-time detection with 1 bot and 6 attack vectors which is presented in section 6.3.