Forensic Science International: Digital Investigation 32 (2020) 300926

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS 2020 EU — Proceedings of the Seventh Annual DFRWS Europe

IoT Botnet Forensics: A Comprehensive Digital Forensic Case Study on Mirai Botnet Servers

Xiaolu Zhang, Oren Upton, Nicole Lang Beebe[*], Kim-Kwang Raymond Choo

Department of Information Systems & Cyber Security, University of Texas at San Antonio, San Antonio, TX, 78249, USA

A R T I C L E I N F O

Article history:

Keywords:
Mirai
IoT malware
Forensics
Botnet server

A B S T R A C T

Internet of Things (IoT) bot malware is relatively new and not yet well understood forensically, despite its potential role in a broad range of malicious cyber activities. For example, it was abused to facilitate the distributed denial of service (DDoS) attack that took down a significant portion of the Internet on October 21, 2016, keeping millions of people from accessing over 1200 websites, including Twitter and NetFlix for nearly an entire day. The widespread adoption of an estimated 50 billion IoT devices, as well as the increasing interconnectivity of those devices to traditional networks, not to mention to one another with the advent of fifth generation (5G) networks, underscore the need for IoT botnet forensics. This study is the first published, comprehensive digital forensic case study on one of the most well known families of IoT bot malware - Mirai. Past research has largely studied the botnet architecture and analyzed the Mirai source code (and that of its variants) through traditional static and dynamic malware analysis means, but has not fully and forensically analyzed infected devices or Mirai network devices. In this paper, we set up a fully functioning Mirai botnet network architecture and conduct a comprehensive forensic analysis on the Mirai botnet server. We discuss forensic artifacts left on the attacker's terminal, command and control (CNC) server, database server, scan receiver and loader, as well as the network packets therefrom. We discuss how a forensic investigator might acquire some of these artifacts remotely, without direct physical access to the botnet server itself. This research provides findings tactically useful to forensic investigators, not only from the perspective of what data can be obtained (e.g., IP addresses of bot members), but also important information about which device they should target for acquisition and investigation to obtain the most investigatively useful information.

1. Introduction

Due to the weak defenses and access protections found in many Internet of Things (IoT) devices, the Mirai botnet has had a widespread, significant impact. The public release of its source code[1] released in 2016 has led to a large number of Mirai variants and increased frequency of Distributed Denial of Service (DDoS) attacks (Antonakakis et al., 2017). Mirai, which means 'future' in Japanese, foreshadowing a more than a one time event, modeled the future of significant attacks to come. Mitigation efforts include patching the vulnerabilities that are leveraged by the Mirai malware family and detecting/preventing Mirai from entering IoT networks.

Many investigations of Mirai to date have focused on a traditional malware analysis of the executable code found on infected IoT devices, which can be collected from an infected device or a honeypot (Kambourakis et al., 2017; Kolias et al., 2017; Margolis et al., 2017b; Wang et al., 2018). However, from a forensic investigator's perspective, the executable for infecting an IoT device is not the only area worth exploring in a Mirai network. It can be much more valuable to investigate the control server or servers, if they can be physically collected or remotely accessed, because they contain key information about the whole botnet, not just a single compromised IoT device.

After Mirai's initial launch, there has been increasing abuse of Mirai's source code. Someone lacking the expertise to write an IoT botnet can easily build their own Mirai botnet for a DDoS attack. In this specific scenario, a forensic investigator might be involved in a case where the control server of a Mirai botnet is captured. Answering the following questions can help determine culpability

* Corresponding author.
E-mail addresses: xiaolu.zhang@utsa.edu (X. Zhang), oren.upton@utsa.edu (O. Upton), nicole.beebe@utsa.edu (N.L. Beebe), raymond.choo@fulbrightmail.org (K.-K.R. Choo).

[1] https://github.com/jgamblin/Mirai-Source-Code

https://doi.org/10.1016/j.fsidi.2020.300926

of the owners of systems hosting the botnet.

- What forensic approaches work on the botnet servers?
- What evidence is retrievable from the servers?
- Where is the evidence located?
- What investigative information can be obtained from the evidence?

In this paper, we propose the first comprehensive digital forensic case study on the server side of a typical Mirai botnet. In this study, existing forensic approaches were applied for data acquisition and analysis. We acquired the disk image, memory (RAM) image, and network traffic (for the attacker's terminal only) from the control servers of a pre-built Mirai botnet, and then conducted manual analysis of the different servers in the Mirai botnet.

The main goal of our study was to recover the list of the infected IoT devices in the botnet, and the historical record for the DDoS attacks launched by the attacker. We also sought to determine how an investigator might deal with a lack of physical access to the Mirai servers. Another objective for this study was to recover as many login credentials as possible, so that investigators could gain remote access to any unobtainable servers. To achieve these goals, we reverse engineered the service executable and live processes that were extracted from the memory and disk images. Additionally, to guide a forensic investigator through such a complex scenario, we summarized our findings in a road map for Mirai botnet server forensics, which clearly shows the relationship between the data source acquired and the evidence that can be recovered.

The rest of this article is structured as follows. In the next section, we briefly introduce the functionality of different components of a typical Mirai botnet. Then, in Sec. 3 we present the high-level methodology of our case study. After setting up the experimental environment in Sec. 4, the case study and the findings (forensic artifacts) are presented in Sec. 5. The findings are then summarized in a decision tree, in Sec. 6. The related literature and the conclusion are presented in Sec. 7 and Sec. 8, respectively.

2. Mirai botnet overview

In this section, we briefly introduce the components underlying a typical Mirai botnet, which are then used to breakdown the DDoS attack and infection process (by which a Mirai botnet can be expanded).

According to the source code of Mirai, the foundation of a typical Mirai botnet consists of a Command & Control (CNC) server, a MySQL database server, a Scan Receiver, a Loading server (or Loader), and a DNS server. The author(s) of Mirai recommended either a ‘trivial’ setup or a ‘professional’ setup,[2] where some services were recommended to be physically added to the same device. As Fig. 1 shows, an attacker can launch a DDoS attack by sending a dedicated command from a Remote Terminal to the CNC server (step *a*) through *Telnet*. Simultaneously, the command is recorded historically on the MySQL database server (step *b*) and then the target of the attack is forwarded to the infected IoT devices (or bots) in step c_1. In response, the bots that are currently alive would follow the CNC's order by sending a flood of network packets to the victim server that is targeted (step d_1).

Additionally, an infected IoT device is capable of exploring the network for other vulnerable IoT devices from a wide range of IP addresses (step *i*). When a vulnerable device is found (‘vulnerable’ here refers to those IoT/Linux devices with weak SSH and Telnet user credentials), the bot would report this finding (including the IP address, user credential, type of service, etc.) to the Scan Receiver (step *ii*). As a new report is received, the vulnerable device's information would be captured by the Loader proactively. As Fig. 1 shows, the Scan Receiver and the Loader were considered here to be on the same machine because, by default, the Scan Receiver would add the vulnerable device's information to the Standard Output stream (or *stdout*) of the Operating System that is always monitored by the Loader (step *iii*).

Next, the Loader would log in to the vulnerable device and upload the malware (step *iv*). After the new IoT device is infected, it will then be set up as a new bot and the new bot must register itself with the CNC server (step *vi*). However, there is one step, that we highlight here, that was neglected by almost all the existing research which is extremely important for forensic investigators. The vulnerable device must retrieve the IP address of the CNC server from a hard-coded DNS server (step *v*), and the same situation happens when an infected device needs to communicate with the Scan Receiver. Because of this design, as long as the DNS server is alive the attacker can move all other servers to a different IP address.

With this overview, we can now more specifically state the scope of this research. This research aims to identify the owner of the devices running the servers in Fig. 1. Forensic analysis of infected IoT devices is out of scope of this research, as is forensic analysis of the victims of the DDoS attack. Rather, we focus on devices under the attacker's control (region ‘A’ and ‘A/B’).

3. Methodology

As previously mentioned, this study focuses on conducting a forensic examination of the server and control side of a typical Mirai botnet. The goal for this study was to discover as many forensic artifacts as possible from the servers that are captured physically (e.g., through a law enforcement seizure) or logically (e.g., through legitimate remote access). Therefore, we will now provide a high-level description of the forensic analysis process:

1. We built our own local Mirai botnet with the open source code on GitHub. This botnet was set up with the exact same network topology shown in Fig. 1.
2. We acquired data from the file system, RAM, and network traffic for each physical server.
3. We manually analyzed the data source acquired in the preceding step. The main goal of the analysis was to find as much evidence as possible regarding: a) the historical record of the achieved attacks (e.g., who, when, and how long the attacks were launched), b) the victim/target of the DDoS attack, and c) the information about the infected bots.
4. Ultimately, we verified the findings with Mirai's source code, and refined the findings. The findings were then incorporated into a Road Map, which can be used for identifying and collecting evidence that can be potentially recovered from the data acquired.

Anyone can make a customized Mirai botnet with the public source code. However, the majority of potential attackers consist of individuals who lack the skill or ability to make fundamental changes to the source code. These individuals are often referred to as ‘script kiddies’. From a forensic investigator's perspective, it is possible that they will encounter a case where the Mirai botnet was set up with the original source code, with the only necessary modifications being key IP addresses and user credentials. Therefore, we emphasize that the scope of this research is effective for a forensic analysis on a Mirai network that shares key code elements of Mirai's original source code.

[2] https://github.com/jgamblin/Mirai-Source-Code/blob/master/ForumPost.md

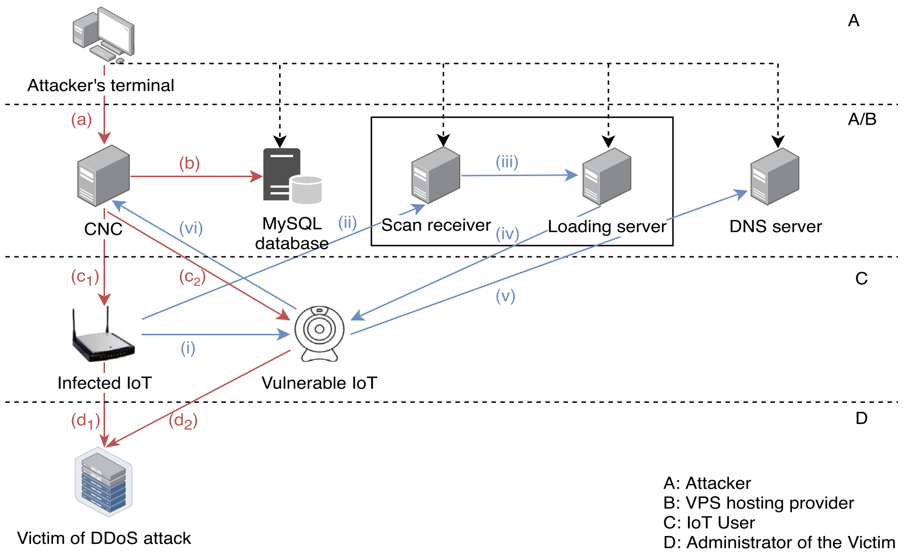


Fig. 1. Mirai botnet topology.

4. Environment setup

To conduct a forensic analysis on a Mirai botnet, we downloaded Mirai's source code from the aforementioned GitHub repository and set up our testing environment with a similar topology shown in Fig. 1. As Table 1 shows, we set up the botnet servers and the IoT devices, as well as the DDoS attacker host and victim host in separate subnetworks `192.168.1.0/24` and `192.168.4.0/24`, respectively. Devices from the two subnetworks can communicate with one another through a router. Since the router is not a required component for a Mirai botnet, we utilized the router as the Terminal as well, on which an attacker can log into the CNC server.

Changes made in the original Mirai source code were minimal, including changing certain key IP addresses, as well as minimal necessary changes to the configuration file of the servers (the corresponding login credentials can be found in Table 1).

- The MySQL database server was configured to support remote access from the CNC server. The login credentials of the CNC server were stored in a table of the database.
- The IP address and login credentials of the MySQL database were hard-coded to the CNC server.
- A bot (executable) was modified to include the IP address of the DNS server. This is hard coded in the bot infected IoT device.
- A weak password was set on the 'root' user of the 'Vulnerable IoT' device, and this password was included in Mirai's password dictionary to ensure the 'Vulnerable IoT' device can be detected by the infected IoT device.

We utilized two Raspberry Pi 3 Model B+ computers to model the 'infected IoT device' and the 'vulnerable IoT device', because this Raspberry Pi Model uses an ARM processor that is typical of IoT devices. We expected the Mirai bot executable to perform in a manner very close to what an investigator may encounter on many other real-world IoT devices. All other servers were standardized to run on x86-64 computers with Debian 10 Operating System using 2 GB of RAM and a 10 GB Hard Disk Drive.

Additionally, we loaded our 64-Bit Kali Linux forensic workstation with a number of forensic tools; some of these were preinstalled but needed to be updated. For example, Linux Memory Extractor (LiME) and Volatility 2.6 were utilized for memory (RAM) acquisition and analysis respectively. We used DD 8.3 for acquiring the disk image, and Autopsy 4.11.0 was installed for file system analysis and data recovery. Wireshark 3.0.3 was used as a network traffic monitor and Packet CAPture (PCAP) file analyzer. Bulk Extractor was utilized for extracting network packets from the memory dump. The National Security Agency (NSA) open source tool – Ghidra 9.0.4 – was selected for reverse engineering the executable files (in this case, the executable files were compiled by *Golang*).

Table 1
IP address & Login credentials on the devices of the experimental Mirai botnet

Purpose	IP address	User name	Password
Router (Terminal)	192.168.1.1192.168.4.1		
CNC	192.168.1.3	'mirai-user'	'mirai-pass'
MySQL	192.168.1.4	'db-login-usrname'	'db-login-passwd'
Scan Receiver	192.168.1.5		
Loader			
DNS	192.168.1.7		
Infected IoT	192.168.4.2		
Vulnerable IoT	192.168.4.3	'root'	'pass'
Victim server	192.168.4.4		

5. Forensic artifacts

5.1. Attacker's terminal

According to Mirai's design, an attacker must access the CNC server through a remote/local *Telnet* connection. When a connection is made, user credentials (stored in the database server) must be verified for logging in. If the login is successful, the CNC server provides a dedicated shell where a certain type of DDoS attack can be issued.

Because the Telnet protocol does not use encryption, a forensic investigator can capture the network packets live or possibly recover the previous network packets from the memory image of the attacker's terminal. The packets would be expected to include the user credentials and the commands typed in through the Terminal. Table 2 shows a comparison table for the CNC shell's legitimate commands and their description. Note that a full command should specify an attack listed in Table 2, a target IP address and its netmask, and a duration time. For example, Terminal command “`udp 192.168.4.4/24 100`” can activate all the live bots to launch UDP floods to the server `192.168.4.4/24` for 100 s.

5.2. CNC server

The CNC server is one of the most forensically valuable servers in the Mirai botnet. As Fig. 1 shows, the CNC server is in charge of issuing attacking orders to the bots, as well as waiting for the newly infected IoT devices to register. We will describe our analysis of the CNC server's disk image and memory image. Specifically, major findings were extracted from: 1) the executable file of the CNC service, 2) the memory data of the CNC process, and 3) the network packets carved from the CNC's memory image.

5.2.1. CNC source code

It is very likely that some easily retrievable and valuable forensic artifacts may be available in the CNC server, which may include the ready-to-compile source code files that the attacker may have failed to remove. In this case, the hard-coded database server's IP address and login credentials can be found in this file `/Mirai-Source-Code/mirai/cnc/main.go` in clear-text, which is shown in Listing 1. Another equally important artifact included in this file is the table name of the Mirai database where the CNC user credentials and the command history are stored. With these findings, one can access the database server and dump the entire database (an example can be found in Sec. 5.3).

```
const DatabaseAddr string = " 192.168.1.4 "
const DatabaseUser string = "db -login - usrname "
const DatabasePass string = "db -login - passwd "
const DatabaseTable string = " mirai "
```

Listing 1: The constant strings of the database server's IP address and login credentials.

5.2.2. CNC executable

Even though these artifacts may be common, it is also common that a forensic investigator may encounter a situation where those artifacts are not available. Therefore, we describe an approach to recover the same information from the CNC's executable file. This is less likely to be removed, because if it is removed, the attacker would need to compile the CNC service whenever the sever reboots.

In Mirai's source code, the CNC server was written with `Golang`. The source code files under `/Mirai-Source-Code/mirai/cnc/` were supposed to be compiled to a single native executable that we named `cnc`. Thus, our goal was to reverse engineer the `cnc` file with Ghidra. As the source code was compiled on a x86-64 computer, the analysis below (with regard to reverse engineering and executable code analysis) will be based on ×64 assembly and 64-bit memory addressing.

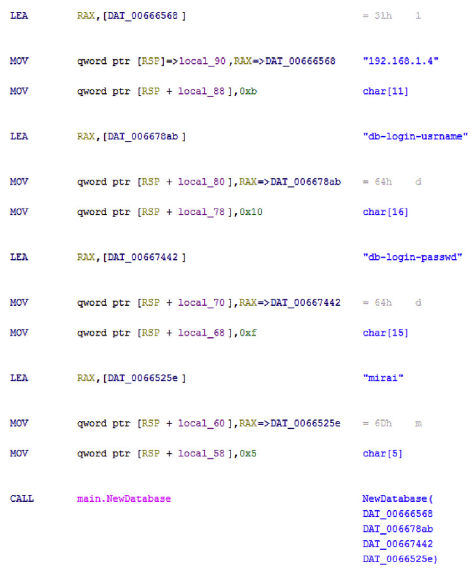
According to the executable analysis, we found that the quickest way to recover the desired information is to locate where the function `NewDatabase(..)` is called in the executable file. This is because the addresses of these four strings in Listing 1 were set as the arguments of this function. As function searching can be easily achieved in Ghidra, Fig. 2 shows the decompiled code snippet where function `NewDatabase(..)` was called. As the figure shows, address `DAT_00666568` (or `0x00666568`) stored the database server's IP address “192.168.1.4”. Likewise, the user name, password and table name can be found at address `0x006678ab`, `0x00667442`, and `0x0066525e` from the `cnc` file, respectively.

5.2.3. CNC live process

Our analysis indicates that the CNC server retains a few queues on the fly, which are used for keeping the active bots, the deleted bots, and the bots carrying out attacks, respectively. The program statement of the queues can be found from the

Table 2
Mirai's terminal commands and description.

ID	Command	Description
N/A	?	Printing the Available Attack List
N/A	botcount	Return the number of live bots
N/A	adduser	add a new CNC user
0x1	vse	Valve source engine specific flood
0x2	dns	DNS resolver flood using the targets domain, input IP is ignored
0x3	syn	SYN flood
0x4	ack	ACK flood
0x5	stomp	TCP stomp flood
0x6	greip	GRE IP flood
0x7	greeth	GRE Ethernet flood
0x9	udpplain	UDP flood with less options, optimized for higher PPS
0x9	udp	UDP flood with more options
0xa	http	HTTP flood



```
LEA     RAX,[DAT_00666568 ]                                 = 31h    1

MOV     qword ptr [RSP]=>local_90 ,RAX=>DAT_00666568        "192.168.1.4"
MOV     qword ptr [RSP + local_88 ],0xb                     char[11]

LEA     RAX,[DAT_006678ab ]                                 "db-login-usrname"

MOV     qword ptr [RSP + local_80 ],RAX=>DAT_006678ab       = 64h    d
MOV     qword ptr [RSP + local_78 ],0x10                    char[16]

LEA     RAX,[DAT_00667442 ]                                 "db-login-passwd"

MOV     qword ptr [RSP + local_70 ],RAX=>DAT_00667442       = 64h    d
MOV     qword ptr [RSP + local_68 ],0xf                     char[15]

LEA     RAX,[DAT_0066525e ]                                 "mirai"

MOV     qword ptr [RSP + local_60 ],RAX=>DAT_0066525e       = 6Dh    m
MOV     qword ptr [RSP + local_58 ],0x5                     char[5]

CALL    main.NewDatabase                                    NewDatabase(
                                                            DAT_00666568
                                                            DAT_006678ab
                                                            DAT_00667442
                                                            DAT_0066525e)
```

Fig. 2. Decomplied CNC executable.

ClientList data structure that is included in source code file/Mirai-Source-Code/mirai/cnc/clientList.go. Therefore, from a forensic investigator's perspective, the data in the queues are the most valuable forensic artifacts for recovering the information about the bots from the CNC server, and the memory dump of the CNC server is the most important, and our analysis proved that this may be the only data source available for recovering the integrated list of bots. Although we argue later in Sec. 5.2.4, that maybe this information regarding the bots can be found from the recovered network packets, this may not always be available.

```
type Bot struct {
uid int
conn net . Conn
version byte
source string
}
```

Listing 2: The statement of the 'Bot' data structure in the source code file bot:go

In order to recover the bots, we targeted the `Bot` data structure (stated in the source code file `/Mirai-Source-Code/mirai/cnc/bot.go`), which is known as a member of the queues and is created for each bot known by the CNC. Listing 2 bot shows the source code statement (in Golang) of 'Bot', which includes a bot's ID ('uid'), an available network connection ('conn'), a version of the network connection ('version') and a string of an IP address ('source'). As Golang is a high level language, this data structure is not documented at the binary level officially. Therefore, we reverse engineered the CNC executable file and explored the memory dump in order to provide the procedure for recovering the bots.

As Fig. 3 shows, a forensic investigator can follow the procedure to find the IP address of the bots from a series of `Bot` data structures. To avoid confusion caused by address translation (from virtual memory address to physical memory address), the addresses shown in Fig. 3 are all virtual memory addresses of the CNC process. If a reader needs to work on the CNC server's memory dump, address translation[3] should be applied additionally.

To retrieve a bot's IP address from the live process, the investigator should first locate a 32-byte data structure (i.e. the binary representation of Listing 2) allocated to the bot. As step [*A*] in Fig. 3 shows, the first 8-byte stores the 'uid' of the bot, the second 8-byte stores the starting address of the "*net*.*Conn* [4]" function (in this case, the starting address was `0x006a1780`), the third 8-byte includes a pointer to step [*B*], and the last 8-byte holds the 'version' of the network which is usually assigned with `0x1` for IPv4. In the context of our study, the best way to find the `Bot` data structure is to confirm the starting address of the "*net*.*Conn*" function (by searching the machine code through the memory) and then utilize the address as a keyword search for the CNC process. Additionally, the investigator should check the data structure located by verifying both 'uid' (the first 8-byte) and 'version' (the last 8-byte).

After the 32-byte data structure is located, in step [*B*], we can find another data structure at `0xc0000b6200`, from which the fixed offsets `0x68` and `0x78` were found storing two pointers. Both pointers are two steps (step [D_1] and [D_2]) further from the IP address of the host (CNC) server and the bot respectively. The binary representation of the host's IP (v) and bot's IP (*iv*) can then be found and converted to `192.168.1.3` and `192.168.4.3`.

5.2.4. *Network packets*

To retrieve as many artifacts as possible, Bulk Extractor was utilized for carving out network packets from the memory dump.

The first type of artifact we found from the packets was related to the previous attacks. The payload highlighted in Fig. 4 shows a typical attacking request sent from the CNC server (`192.168.1.3:23`) to the Infected IoT device (`192.168.4.2:52984`). When the packet was received, the bot would launch an http flood (DDoS) attack which was specified by the attack ID `0x0a` (see Table 2 for more attack IDs), or the seventh byte of the payload. Besides, bytes 9 to 13 (`0xc0 0xa8 0x04 0x04 0x18`) refer to the victim server's IP address `192.168.4.4/24`. Bytes 3 to 6 (`0x00 0x00 0x00 0x63`) specified the duration of the attack which is 99 s in decimal.

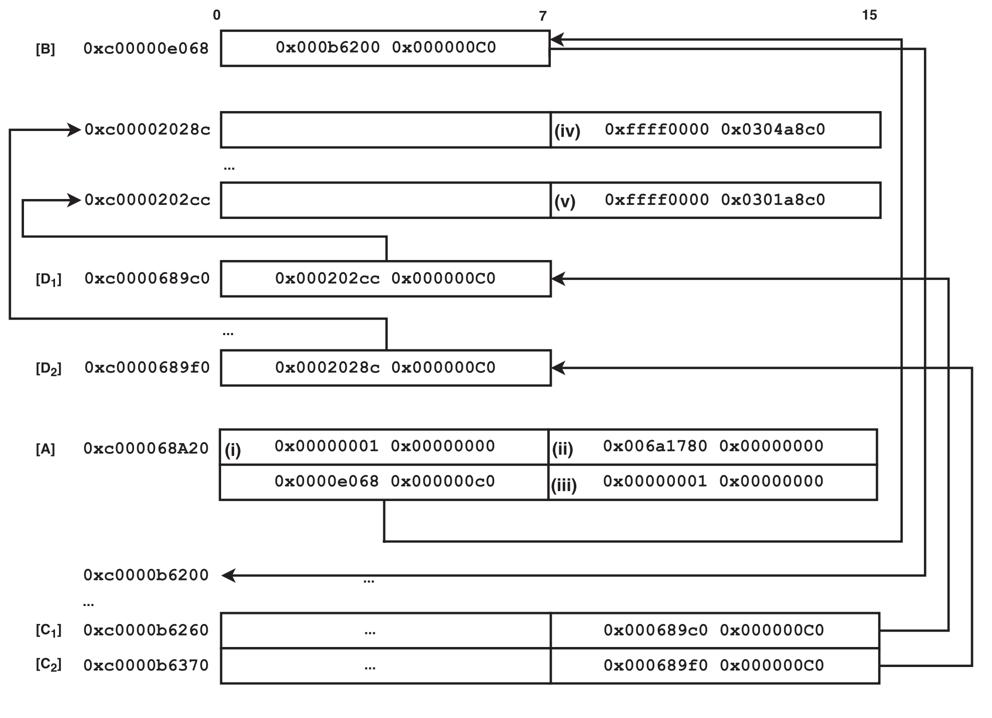
In addition to the "binary" version of attack command, the network packets of the original clear-text commands were found as well, since an attacker is supposed to use a Terminal to make a Telnet connection with the CNC server. Doing so causes the corresponding network packets to be cached in the memory. However, we noticed that a command can be only transferred one character per packet. For example, to recover the full command "`http 192.168.4.4/24 99`" a series of 22 Telnet packets (including space) must be found. Fig. 5 shows the second packet of this command, which only carries the second character 't' in 'http' rather than the whole command.

Second, we found the user credentials transferred in a similar format as the "plain-text" command, thus these were found in single character from the recovered network packets as well. Fig. 6 shows the TCP stream of the Telnet packets, which includes the CNC user name "mirai-user", the password "mirai-pass" and three standard CNC commands: "?" (output CNC's help menu), "botcount" (counting the number of bots) and "udp 192.168.4.4/24 99" (launching UDP flood attack).

The other type of packet we found that carried recoverable CNC's user credentials was the packet sent between the CNC and

[3] The memmap command of Volatility can provide a map of virtual page and physical page.

[4] Function *net*.*Conn* is part of the public Golang library *net* that provides a portable interface for network I/O.



(i) UID
(ii) The address of Golang function "go.itab.* net.TCPConn,net.Conn".
(iii) Version
(iv) Bot's IP 192.168.4.3
(v) Host IP 192.168.1.3

Fig. 3. Bot list acquisition from CNC live memory: An example.

the MySQL server. Instead of one character per packet, the MySQL protocol as Fig. 7 shows used a single packet to transfer the user credentials.[5]

Third, since bots must contact the CNC server proactively, they were designed to send a registration packet when they join the botnet the first time, and keep sending pulse packets to state their activeness. The packets in Fig. 8 and Fig. 9 are the samples of a registration packet and a pulse packet respectively. Typically, the payload of a registration packet must be `0x00 0x00 0x00 0x01` and the payload of a pulse packet must be `0x00 0x00`. By recognizing these packets, a forensic investigator might be able to recover a list of the active bots (including the bot's IP address, and time for their first/last contact, etc.).

5.3. Database server

As Fig. 1 shows, the MySQL database server is supposed to only interact with the CNC server. In Mirai's source code package, the Script file "`/Mirai-Source-Code/scripts/db.sql`" was used for creating a 'mirai' database including 1) a 'history' table that stores the history commands issued on the CNC server, 2) a 'users' table that contains CNC login credentials, and 3) a 'whitelist' table that retains the IP addresses that the bots are not allowed to scan. Because one can access the database server legitimately by leveraging the user credentials retrieved from the CNC server (see Sec. 5.2), we conducted our forensic examination through both physical and remote access.

Once a forensic investigator gains physical access to a Mirai database server, taking a physical image of the server is always the most effective option for the following analysis, in which the deleted database files can be possibly recovered on disk. To retrieve evidence from the 'mirai' database, one should pay attention to recover `.idb` files and `.frm` files from the default 'mirai' database folder `/var/lib/mysql/mirai/`, since if the setting `innodb_file_per_table`[6] were enabled MySQL would generate these two files (named `<table name>.idb` and `<table name>.frm`) for each database table. Indeed, due to a different database configuration, the location of these files could vary. Thus, a forensic investigator should recover and check the configuration file to confirm the data location of the MySQL database. Fig. 10 shows the sample artifacts retrieved from the MySQL database.

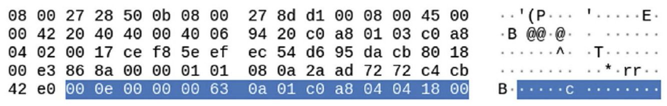
```
mysqldump -h 192.168.1.4 -P 3306 -u db-login-usrname -p
db-login-passwd mirai
```

Listing 3: Sample command for dumping the mirai database remotely.

If the ivestigator does not have physical access to the host of the

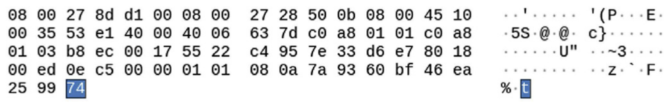
[5] The reason MySQL uses one packet to carry the user credentials is that the CNC server verified the user credentials with a single SQL query — "SELECT username, max_bots, admin FROM users WHERE username = ? AND password = ?". In terms of the MySQL protocol, the SQL statement and its parameters ("mirai-user" and "mirai-pass") must be transferred in two independent packets.

[6] Database is stored in a single file if innodb_file_per_table were not enabled.



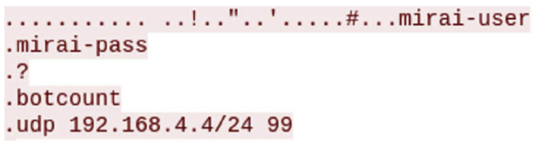
```
08 00 27 28 50 0b 08 00  27 8d d1 00 08 00 45 00   ··'(P··· '·····E·
00 42 20 40 40 00 40 06  94 20 c0 a8 01 03 c0 a8   ·B @@·@· · ······
04 02 00 17 ce f8 5e ef  ec 54 d6 95 da cb 80 18   ······^· ·T······
00 e3 86 8a 00 00 01 01  08 0a 2a ad 72 72 c4 cb   ········ ··*·rr··
42 e0 00 0e 00 00 00 63  0a 01 c0 a8 04 04 18 00   B······c ········
```

Fig. 4. Sample packet including the “binary” version of a command that launched an http flood attack.



```
08 00 27 8d d1 00 08 00  27 28 50 0b 08 00 45 10   ··'····· '(P···E·
00 35 53 e1 40 00 40 06  63 7d c0 a8 01 01 c0 a8   ·5S·@·@· c}······
01 03 b8 ec 00 17 55 22  c4 95 7e 33 d6 e7 80 18   ······U" ··~3····
00 ed 0e c5 00 00 01 01  08 0a 7a 93 60 bf 46 ea   ········ ··z·`·F·
25 99 74                                           %·t
```

Fig. 5. Sample packet including the “text” version of a command that launched an http flood attack.



```
........... ..!..".."'.....#...mirai-user
.mirai-pass
.?
.botcount
.udp 192.168.4.4/24 99
```

Fig. 6. TCP stream of Telnet packets including CNC user credentials and legitimate CNC commands.

MySQL database, but they have a legitimate remote access to the MySQL database, the tables can be dumped from the database server by using tool `mysqldump`. For example, with the recovered user name ‘db-login-usrname’ and the password ‘db-login-passwd’, one can dump the entire ‘mirai’ database remotely from 192.168.1.4:3306 by using the ‘mysqldump’ command in Listing 3.

5.4. *Scan Receiver & Loader*

If the Scan Receiver and Loader server is the only host to which an investigator has gained physical access, it is still possible to recover substantial information regarding the bot list and the attack history from the memory & the disk image.

5.4.1. *Network packets*

Through our examination, network packets were extracted from the server's memory image, which included the information of the infected/vulnerable IoT devices. For example, Fig. 11 shows a packet carved from the memory image. The packet was sent to the Scan Receiver by an infected IoT device. The packet includes the IP address of a vulnerable IoT device and its ‘weak’ login credentials that have been tested for an unauthorized login. To parse the payload of the packet (highlighted from the figure), we found that “`0xC0 0xA8 0x04 0x03 0x00 0x17`” refers to the vulnerable device's IP address “`192.168.4.3`” and the port ‘23’ (Telnet), “`0x72 0x6f 0x6f 0x74 0x04 0x70 0x61 0x73 0x73`” refer to the ASCII code of login user name ‘root’ and password ‘pass’, and these sections were separated with their length ‘`0x04`’.

5.4.2. *Standard output stream*

The standard output stream (or ‘stdout’) is where the Loader acquires the information of a vulnerable IoT device reported by a bot. By default, when the information was received by the Scan Receiver it is posted on the ‘stdout’ in plain-text. As the data posted on ‘stdout’ is not supposed to be stored permanently, we argue that the memory image should be the first priority for forensic analysis. Specifically, the easiest way to recover the information of vulnerable IoT devices is to search for and analyze the specific format of these records, in which, as the data highlighted in Fig. 12 shows, a record must be structured with an IP address, a space (0x20), a string of user name, a colon, a string of password and a new line mark (0 $\times$ 0A). Thus, we can determine that there were two vulnerable devices reported to the Scan Receiver. Both of the records indicate to the same vulnerable device at IP address 192.168.4.3 with user name ‘root’ and password ‘pass’ (however, the IP address of the reporting IoT devices were not included as they lack importance for the Loader).

Although the data of the ‘stdout’ can be located in a more normal way (such as tracking the “Terminal” process from the memory image in which the ‘stdout's file descriptor was opened), we found through our case study that keyword searching may work more efficiently than any other approaches, because the format of the record seems unlikely to be changed. This is likely because otherwise the Loader's source code must be changed for parsing the record. Another key point for forensic investigators to note is that these records might be visible on the screen of the server when they are captured. In this case, an investigator should acquire the records as well as the format for later keyword searching.

5.4.3. *Malicious/bot executable*

Since a Loader must store bot executables (in different architectures) for infecting the vulnerable IoT devices, these bot executables can be recovered from the server's disk image. As Fig. 1 shows, a malicious executable must hard-code the IP address of the DNS server in order to find the CNC server and the Scan Receiver's IP address. Therefore, we argue that the malicious executable can be utilized as a probe of the botnet. Running the executable in a Sandbox can help us monitor/collect information related to future attacks using this botnet.

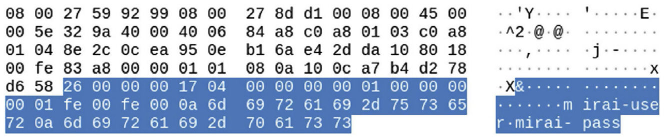
5.5. *DNS server*

There are many different ways to set up Mirai's DNS, as it is only used for forwarding the CNC and the Scan Receiver's IP address to bots. In this section, the investigation was demonstrated on *BIND 9*, which is one of the most commonly used DNS servers in Linux. Unsurprisingly, we recovered the CNC server and the Scan Receiver's IP address and the client (bot) list by verifying those who had ever requested the CNC server and the Scan Receiver's IP address.

```
$TTL 604800
$ORIGIN mirai . com
@ IN SOA mirai . com . root . mirai . com .(
1 ; Serial
604800 ; Refresh
86400 ; Retry
2419200 ; Expire
604800) ; Negative Cache TTL
;
www IN A 192.168.1.3
report IN A 192.168.1.5
```

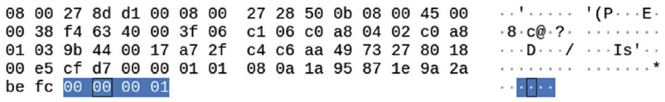
Listing 4: The recovered comparison table of Domain name and IP address.

BIND 9 is supposed to keep the comparison table of Domain name and IP address (or DNS table) in a configuration file. To track this file, one can start from the primary configuration file `/etc/bind/named.conf`, in which the secondary configuration files are



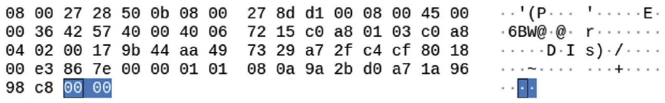
```
08 00 27 59 92 99 08 00  27 8d d1 00 08 00 45 00   ··'Y····  '·····E·
00 5e 32 9a 40 00 40 06  84 a8 c0 a8 01 03 c0 a8   ·^2·@·@·  ········
01 04 8e 2c 0c ea 95 0e  b1 6a e4 2d da 10 80 18   ···,····  ·j·-····
00 fe 83 a8 00 00 01 01  08 0a 10 0c a7 b4 d2 78   ········  ·······x
d6 58 26 00 00 00 17 04  00 00 00 00 01 00 00 00   ·X&·····  ········
00 01 fe 00 fe 00 0a 6d  69 72 61 69 2d 75 73 65   ·······m  irai-use
72 0a 6d 69 72 61 69 2d  70 61 73 73               r·mirai-  pass
```

Fig. 7. A sample MySQL packet carrying CNC user credentials.



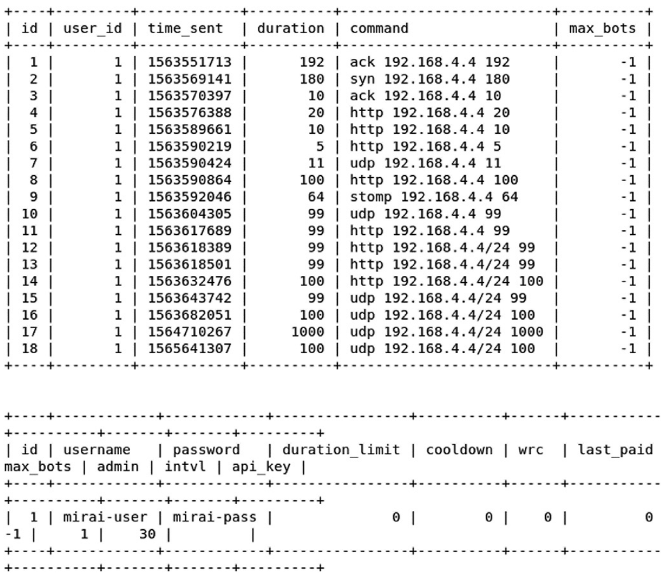
```
08 00 27 8d d1 00 08 00  27 28 50 0b 08 00 45 00   ··'·····  '(P···E·
00 38 f4 63 40 00 3f 06  c1 06 c0 a8 04 02 c0 a8   ·8·c@·?·  ········
01 03 9b 44 00 17 a7 2f  c4 c6 aa 49 73 27 80 18   ···D···/  ···Is'··
00 e5 cf d7 00 00 01 01  08 0a 1a 95 87 1e 9a 2a   ········  ·······*
be fc 00 00 00 01                                  ······
```

Fig. 8. A sample registration packet from a bot.



```
08 00 27 28 50 0b 08 00  27 8d d1 00 08 00 45 00   ··'(P···  '·····E·
00 36 42 57 40 00 40 06  72 15 c0 a8 01 03 c0 a8   ·6BW@·@·  r·······
04 02 00 17 9b 44 aa 49  73 29 a7 2f c4 cf 80 18   ·····D·I  s)·/····
00 e3 86 7e 00 00 01 01  08 0a 9a 2b d0 a7 1a 96   ···~····  ····+····
98 c8 00 00                                        ····
```

Fig. 9. A sample pulse packet from an active bot.



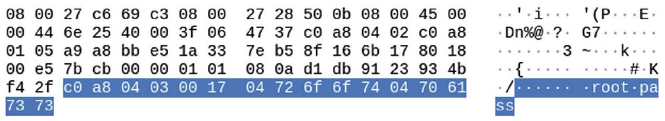
```
+----+---------+------------+----------+--------------------------+----------+
| id | user_id | time_sent  | duration | command                  | max_bots |
+----+---------+------------+----------+--------------------------+----------+
|  1 |       1 | 1563551713 |      192 | ack 192.168.4.4 192      |       -1 |
|  2 |       1 | 1563569141 |      180 | syn 192.168.4.4 180      |       -1 |
|  3 |       1 | 1563570397 |       10 | ack 192.168.4.4 10       |       -1 |
|  4 |       1 | 1563576388 |       20 | http 192.168.4.4 20      |       -1 |
|  5 |       1 | 1563589661 |       10 | http 192.168.4.4 10      |       -1 |
|  6 |       1 | 1563590219 |        5 | http 192.168.4.4 5       |       -1 |
|  7 |       1 | 1563590424 |       11 | udp 192.168.4.4 11       |       -1 |
|  8 |       1 | 1563590864 |      100 | http 192.168.4.4 100     |       -1 |
|  9 |       1 | 1563592046 |       64 | stomp 192.168.4.4 64     |       -1 |
| 10 |       1 | 1563604305 |       99 | udp 192.168.4.4 99       |       -1 |
| 11 |       1 | 1563617689 |       99 | http 192.168.4.4 99      |       -1 |
| 12 |       1 | 1563618389 |       99 | http 192.168.4.4/24 99   |       -1 |
| 13 |       1 | 1563618501 |       99 | http 192.168.4.4/24 99   |       -1 |
| 14 |       1 | 1563632476 |      100 | http 192.168.4.4/24 100  |       -1 |
| 15 |       1 | 1563643742 |       99 | udp 192.168.4.4/24 99    |       -1 |
| 16 |       1 | 1563682051 |      100 | udp 192.168.4.4/24 100   |       -1 |
| 17 |       1 | 1564710267 |     1000 | udp 192.168.4.4/24 1000  |       -1 |
| 18 |       1 | 1565641307 |      100 | udp 192.168.4.4/24 100   |       -1 |
+----+---------+------------+----------+--------------------------+----------+


+----+------------+------------+----------------+----------+-----+-----------
+----------+-------+-------+---------+
| id | username   | password   | duration_limit | cooldown | wrc | last_paid
max_bots | admin | intvl | api_key |
+----+------------+------------+----------------+----------+-----+-----------
+----------+-------+-------+---------+
|  1 | mirai-user | mirai-pass |              0 |        0 |   0 |         0
-1 |     1 |    30 |         |
+----+------------+------------+----------------+----------+-----+-----------
+----------+-------+-------+---------+
```

Fig. 10. CNC Command history and CNC User credentials recovered from MySQL database server.

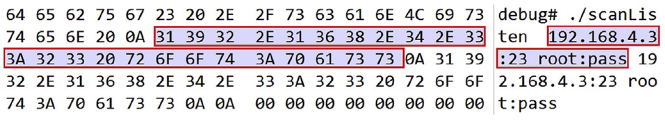
included. Typically, BIND 9 recommends the user add a new zone to the file/etc/bind/named.conf.local to indicate the file that stores the DNS table. By recovering the file `named.conf.local`, the DNS table file can be found, and the DNS table can be recovered. The DNS table recovered from the DNS server is shown in Listing 4, of which the CNC's domain name `www.mirai.com` is corresponding to the IP address `192.168.1.3` and the Scan Receiver's domain name `report.mirai.com` refers to the IP address 192.168.1.5.

Since the DNS server might log the DNS queries, the sample history queries we found from the DNS server's disk image are shown in Fig. 13. Again, please note that the proposed forensic analysis on the DNS is not universal. Neither the log files nor the



```
08 00 27 c6 69 c3 08 00  27 28 50 0b 08 00 45 00   ··'·i···  '(P···E·
00 44 6e 25 40 00 3f 06  47 37 c0 a8 04 02 c0 a8   ·Dn%@·?·  G7······
01 05 a9 a8 bb e5 1a 33  7e b5 8f 16 6b 17 80 18   ·······3  ~···k···
00 e5 7b cb 00 00 01 01  08 0a d1 db 91 23 93 4b   ··{·····  ·····#·K
f4 2f c0 a8 04 03 00 17  04 72 6f 6f 74 04 70 61   ·/······  ·root·pa
73 73                                              ss
```

Fig. 11. The network packet reported from a bot to the Scan Receiver, which includes the IP address and the user credentials of a vulnerable IoT device.



```
64 65 62 75 67 23 20 2E  2F 73 63 61 6E 4C 69 73 debug# ./scanLis
74 65 6E 20 0A 31 39 32  2E 31 36 38 2E 34 2E 33 ten  192.168.4.3
3A 32 33 20 72 6F 6F 74  3A 70 61 73 73 0A 31 39 :23 root:pass 19
32 2E 31 36 38 2E 34 2E  33 3A 32 33 20 72 6F 6F 2.168.4.3:23 roo
74 3A 70 61 73 73 0A 0A  00 00 00 00 00 00 00 00 t:pass
```

Fig. 12. The stdout in the memory which includes the IP address, user name and password of a vulnerable IoT device, which was received by the Scan Receiver and was sent by an infected IoT device.

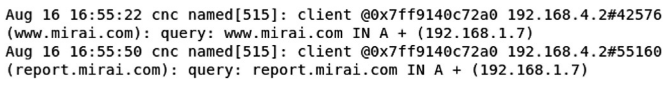
network packets are guaranteed for recovering the full list of clients because when the logs and the packets reach the size limit the oldest data would be lost.

6. A road map for Mirai botnet server forensics

In this section, we propose a road map as a guide for Mirai botnet server forensics, since configurations and deployments can vary. As Fig. 14 shows, analysis of each different server can result in gaining different forensic artifacts, which are considered in two categories. A list of bots can indicate the infected/vulnerable IoT devices under the attacker's control. The command and attack history can prove that the attacker had launched DDoS attacks through the botnet.

To utilize the road map, an investigator can start a forensic analysis from any of the five servers with physical access. The road map indicates the path from a data entity, which can be extracted from the data resource (e.g., Network packets, executable file or live process) to the key evidence. The diamond arrows in the figure indicate that we had proposed an approach to recover the integrated data of the evidence from the data entity. The classic arrows indicate that the evidence may be recovered partially but without a guarantee. For example, the CNC live process that was extracted from the CNC server's memory image is the only data entity from which a full list of live bots can be extracted. Similarly, the database file on the database server is the only data entity where the complete attack (command) history can be retrieved.

In summary, we argue that, the CNC server or the MySQL server should be considered the first priority during Mirai server forensics because it is almost guaranteed to find the login credentials for other servers, Therefore, this will allow recovery of the integrated bot list and the attack history. The second priority is the Scan Receiver & Loader, in which a bot executable file and the cached ‘stdout’ data stream and network packets might expose the bot list and attack history (though the comprehensive data integrity cannot be guaranteed). The servers both considered having the least priority are the attacker's remote terminal



```
Aug 16 16:55:22 cnc named[515]: client @0x7ff9140c72a0 192.168.4.2#42576
(www.mirai.com): query: www.mirai.com IN A + (192.168.1.7)
Aug 16 16:55:50 cnc named[515]: client @0x7ff9140c72a0 192.168.4.2#55160
(report.mirai.com): query: report.mirai.com IN A + (192.168.1.7)
```

Fig. 13. The log file records of the BIND 9 DNS server.

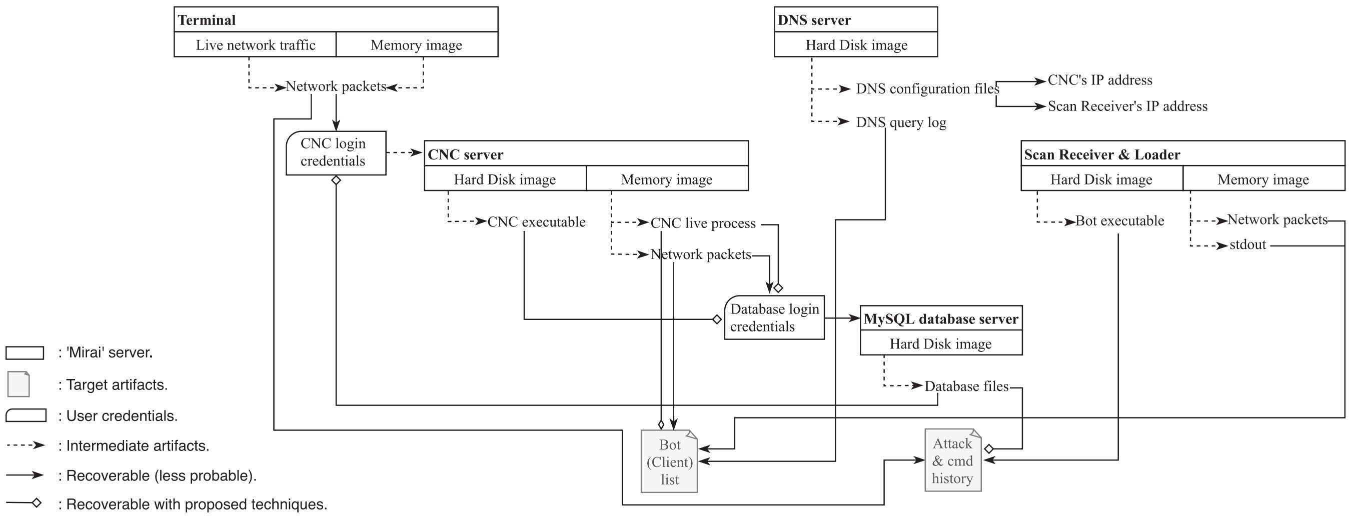


Fig. 14. The road map for Mirai botnet server forensics.

and the DNS server, because logging in the CNC server can barely help an investigator to retrieve the desired evidence as there is no valid command in the shell provided by the CNC that can dump the Mirai database. From a forensic investigator's perspective, the only useful data on these servers are the DNS query log and the terminal's network packets, which can be examined for extracting the information of the bots and the history attack commands respectively although data integrity cannot be guaranteed.

7. Related literature

Mirai was first released in initial attacks in August 2016 and brought widespread attention to weak security common to many IoT devices (Margolis et al., 2017a; Bursztein, 2017). Significant research has since resulted, however, it has largely focused on the executable itself and its variants (Antonakakis et al., 2017; Sinanović and Mrdovic, 2017), as well as the associated botnet architectures (Kolias et al., 2017). We will review that literature briefly, but remind the reader that our focus is on the forensic artifacts associated with the Mirai server. This focus fills a valuable gap for digital forensic practitioners, giving them a comprehensive forensic guide to collecting the most important digital artifacts and data from Mirai servers and its variants.

Kolias et al. (2017) described the Mirai botnet and its variants, including Hajime and Bricker, and described how each botnet worked relative to the other. Their work described the behavior of the botnet at an internet scale, but the work did not describe technical details of how the malware worked inside the CNC servers. Bertino and Islam (2017) also described these and other Internet worms, and described some security measures. However, this work did not describe the technical aspects of the server or the botnet itself. In a more general sense (i.e., not specific to Mirai), Costin and Zaddach (2018) surveyed the existing IoT malware literature.

Kambourakis et al. (2017) performed an in-depth executable analysis, particularly the steps Mirai uses to identify vulnerable systems and how it fingerprints the system. This work further compares Mirai with other malware including Hajime. This technical detail helps researchers understand key elements of how the code works during initial infection and does identify some limited forensic artifacts on victim systems. Sinanović and Mrdovic (2017) analyzed the publicly available Mirai source code using static and dynamic analysis techniques. They describe the general CNC and botnet architecture and how to find some important examples of forensic artifacts, but they do not provide a comprehensive forensic analysis of the CNC server. Similarly, Wang et al. (2017) provided technical details of Mirai and the Darlloz worm with some details useful for forensic analysis, but they too did not provide a comprehensive forensic analysis.

Zulkipli et al. (2017) reiterated the need for forensic understanding of IoT malware. They reviewed the IoT forensic literature, and discussed the commonalities and differences between traditional device forensics and IoT device forensics. They explained that previous digital forensic frameworks either have significant gaps when applied to IoT devices, or the forensic analysis papers they reviewed do not cover key technical details needed. They explained that this is largely due to the heterogeneity of IoT devices, both in hardware and software, as well as other aspects of IoT devices, like the lack of logs and storage, and the inability to easily turn off the device for analysis.

Karabiyik and Akkaya (2019) described general methods for conducting forensic analysis at the network and device level. They also provided an in-depth description of the various communication channels used by IoT devices and the different processors, operating systems, and file systems that complicate IoT forensics. A number of researchers have also forensically examined different IoT devices (Li et al., 2019; Zhang et al., 2019; Chung et al., 2017; Sayakkara et al., 2019) and other IoT malware botnets (Herwig et al., 2019; Soltan et al., 2018), as well as presenting common taxonomy for IoT malware analysis to facilitate the understanding of IoT malware (Soliman et al., 2017). While these works described the behavior of the IoT malware, technical details of how the bot malware worked inside the CNC servers is missing. There remains a lack of comprehensive forensic analyses of botnet servers and devices, including Mirai.

8. Conclusion

With the increasing ubiquity of IoT devices in our society, the advent of IoT bot malware and botnets, combined with 5G networks connecting it all in the very near future, the ability to forensically analyze IoT malware related devices is critical. The impact of the Mirai botnet on the Internet in 2016, as well as the impact of variants thereof since, makes Mirai a great candidate to begin this pursuit. Since 2016, related research has considered the executable itself and the architecture of the botnet that delivers malware and communicates with infected victims. However, the research gap remaining that this research addresses, is the comprehensive forensic analysis of botnet devices — specifically, the Mirai servers. This paper discussed forensic techniques to examine the Mirai botnet server, both remotely and with physical access to it, and outlines specific forensic artifacts and their location. These artifacts provide critical investigative information, including IP addresses and user credentials, among others. This research was limited to the standard, original Mirai botnet set-up. The impacts of significant customization of the botnet was not studied, and might be a topic for future research, although such customizations may be numerous and difficult to predict.

Acknowledgements

This research was supported in part by National Science Foundation CREST under Grant HRD-1736209. UTSA also gratefully acknowledges CACI, Inc. - Federal, for their support of research activities at The Cyber Center for Security and Analytics, UTSA.

References

Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Invernizzi, L., Kallitsis, M., et al., 2017. Understanding the Mirai botnet. In: 26th USENIX Security Symposium, vol. 17. USENIX Security, pp. 1093—1110.

Bertino, E., Islam, N., 2017. Botnets and Internet of Things Security. Computer, pp. 76—79.

Bursztein, E., 2017. Inside Mirai the infamous iot botnet: a retrospective analysis.. https://www.elie.net/blog/security/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis.

Chung, H., Park, J., Lee, S., 2017. Digital forensic approaches for amazon alexa ecosystem. Digit. Invest. 22, S15—S25.

Costin, A., Zaddach, J., 2018. IoT Malware: Comprehensive Survey, Analysis Framework and Case Studies. BlackHat USA.

Herwig, S., Harvey, K., Hughey, G., Roberts, R., Levin, D., 2019. Measurement and Analysis of Hajime, a Peer-To-Peer Iot Botnet. NDSS.

Kambourakis, G., Kolias, C., Stavrou, A., 2017. The Mirai botnet and the IoT zombie armies. In: MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM). IEEE, pp. 267—272.

Karabiyik, U., Akkaya, K., 2019. Digital forensics for IoT and wsns. In: Mission-oriented Sensor Networks and Systems: Art and Science. Springer, pp. 171—207.

Kolias, C., Kambourakis, G., Stavrou, A., Voas, J., 2017. Ddos in the IoT: Mirai and other botnets. Computer 50, 80—84.

Li, S., Choo, K.-K.R., Sun, Q., Buchanan, W.J., Cao, J., 2019. IoT forensics: amazon echo as a use case. IEEE Internet of Things J. 6, 6487—6497.

Margolis, J., Oh, T.T., Jadhav, S., Jeong, J.P., Kim, Y.H., Kim, J.N., 2017a. Analysis and impact of IoT malware. In: Proceedings of the 18th Annual Conference on Information Technology Education. ACM, 187—187.

Margolis, J., Oh, T.T., Jadhav, S., Kim, Y.H., Kim, J.N., 2017b. An in-depth analysis of the Mirai botnet. In: *2017* International Conference on Software Security and Assurance (ICSSA). IEEE, pp. 6—12.

Sayakkara, A., Le-Khac, N.-A., Scanlon, M., 2019. Leveraging Electromagnetic Side-Channel Analysis for the Investigation of Iot Devices arXiv preprint. arXiv:1904.02089.

Sinanović, H., Mrdovic, S., 2017. Analysis of Mirai malicious software. In: 2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM). IEEE, pp. 1—5.

Soliman, S.W., Sobh, M.A., Bahaa-Eldin, A.M., 2017. Taxonomy of malware analysis in the IoT. In: 2017 12th International Conference on Computer Engineering and Systems (ICCES). IEEE, pp. 519—529.

Soltan, S., Mittal, P., Poor, H.V., 2018. Blackiot: iot botnet of high wattage devices can disrupt the power grid. In: 27th USENIX Security Symposium, vol. 18. USENIX Security, pp. 15—32.

Wang, A., Chang, W., Chen, S., Mohaisen, A., 2018. Delving into internet ddos attacks by botnets: characterization and analysis. IEEE/ACM Trans. Netw. 26, 2843—2855.

Wang, A., Liang, R., Liu, X., Zhang, Y., Chen, K., Li, J., 2017. An inside look at IoT malware. In: International Conference on Industrial IoT Technologies and Applications. Springer, pp. 176—186.

Zhang, X., Choo, K.-K.R., Beebe, N.L., 2019. How do i share my iot forensic experience with the broader community? an automated knowledge sharing iot forensic platform. IEEE Internet of Things J. 6, 6850—6861.

Zulkipli, N.H.N., Alenezi, A., Wills, G.B., 2017. IoT forensic: bridging the challenges in digital forensic and the internet of things. In: International Conference on Internet of Things, Big Data and Security, vol. 2. SCITEPRESS, pp. 315—324.