# Analysis of Mirai Malicious Software

Hamdija Sinanović, Sasa Mrdovic
Faculty of Electrical Engineering
University of Sarajevo
Sarajevo, Bosnia and Herzegovina
{hsinanovic1, sasa.mrdovic}@etf.unsa.ba

***Abstract*—This paper tries to shed more light on Mirai malware, with an aim to facilitate its easier detection and prevention. This malware was used in several recent high profile DDoS attacks. Mirai is used to create and control botnet of IoT devices. The code of this malware is analysed and explanation of its parts provided. Virtual environment for dynamic analysis of Mirai is created. Special settings that were needed to install, start and use Mirai in this environment are explained. Mirai CNC user environment with list of commands is presented. Controlled DDoS attack was successfully executed. Traffic generated during controlled attacks was used to generate signature for Mirai detection. Conclusion of static and dynamic analysis is given together with some mitigation advices.**

***Keywords*—Mirai, malware, Internet of Things, Botnet, Distributed Denial of Service**

## I. Introduction

Mirai is malicious software that creates botnet of IoT devices. It drew public attention in September 2016 after it was used in DDoS attack against Kerbs On Security website which reached 620 Gbps [1]. After that, it was used in attack on French hosting company OVH that peaked at 1 Tbps [2]. However, the most prominent Mirai DDoS attack was on DNS provider Dyn, resulting in inaccessibility of several high-profile websites such as GitHub, Twitter, Reddit, Netflix and many others. After analysis, Dyn estimated that there were up to 100 000 malicious endpoints involved in the attack [3].

Mirai was an example on how big IoT insecurity problem might be. It was an eye opener. Malicious software exists for almost as long as a regular software. Before IoT it was mainly limited to computers. Computer security community developed methods and tools to fight malicious software. IoT devices do not have full computing abilities that anti malware methods and tools rely on. They are low price devices usually built for one purpose with ability to communicate over network. This ability opens an avenue for attacks. Due to lack of convenient user interface, IoT devices often have default user names and passwords. In addition they are not envisioned to receive software updates, including security ones.

Mirai successfully abused IoT device properties. Analysis of Mirai enables researchers to learn how it works. That knowledge helps to detect and protect from the similar attacks in the future.

In this paper we show results of static and dynamic analysis of Mirai. We explain the function of various parts of its code and how all the parts fit together. We have created controlled virtual environment, that enabled us to see Mirai in action. We explain steps we had to do in order to run Mirai in this environment. We were able to gain some additional knowledge that enabled us to create IDS rules for its detection. We share our findings in belief it will help other security and IoT researchers.

The rest of the paper is organized as follows.

Related work is addressed in section 2. Section 3 describes our static analysis and its results. Dynamic analysis is presented in section 4. Proposal for Mirai detection based on previous findings is provided in section 5. Short overview of our observations and some mitigation ideas are given in Section 6. Conclusion and discussion, as well as directions for future research work, are in section 7.

## II. Related work

Malware analysis is similar to any software analysis. In the beginning it was mainly static analysis [4][5][6]. It is useful, but has its limitations. The biggest limitation is inability to guarantee that a piece of code is not malicious. It has been shown, a long time ago that no amount of static analysis can do that [7][8].

Dynamic code analysis is performed by executing programs in real or, more often, virtual environment. It provides different insight into malware behaviour. Automated dynamic analysis in a sandboxed environment like the one described in [9], seems the most promising. The good overview of automated dynamic malware tools and techniques is given in [10].

IoT malware is rather recent. Most of the available literature on IoT malware comes from non-scientific web sites and antivirus software makers. There are some papers that were published on recent conferences. An overview of issues in IoT security with a proposal for research for the solutions to the IoT security challenges is given in [11]. In [12] authors created IoT honeypot that enabled them to analyze telnet based attacks. Their honeypot attracted various attack on IoT devices running on different CPU architectures.

Mirai attack attracted a lot of attention that resulted in a number of publications. Out of non-scientific web articles we would like to point out [13] that includes some Mirai static code analysis partly based on [14]. Recent issue of IEEE Computer has an article on botnets and IoT security that includes some general Mirai analysis [15]. This years RSA conference had a presentation with detailed analysis of Mirai, but only slides are available [15]. Author of [16] provides thorough analysis of IoT botnets that includes Mirai. The most

detailed analysis we found is part of master thesis project [17]. It includes source code analysis and some experiments.

We believe that there is a need for more Mirai analysis and provide our contribution in the rest of the paper.

## III. Static analysis

Luckily, Mirai's source code was leaked for unknown reasons, making static analysis reasonably easy [18]. First thing to be noticed is a build script, which compiles bot source code for ten different architectures. It can also be noticed that source code is divided in three parts: `bot`, `CNC server` and `loader`. `Bot` part runs on infected IoT devices. `CNC server` receives connections from `bots` and issues commands to them. `Loader` receives information on discovered vulnerable IoT devices and serves them `bot` payload for their architecture.

### A. Bot

`Bot` was written entirely in C programming language. Starting at `main.c` file, it can be seen that Mirai deletes it's exe file once it's started, staying only in RAM. It is one of its ways for avoiding detection. Since persistence is not ensured, malware disables watchdog timer on infected device, preventing it from restarting. Then it checks for and kills another instance of the same malware already running on the same device. Random name for it's process is generated to make detection harder. After that, it calls `fork()` system call multiple times to create process for each module. Then it connects to the CNC server and waits for commands to be executed. There are three modules running beside main process: `attack`, `killer` and `scanner`.

*1) Attack:* `Attack` module parses command when received and launches DoS attack. Ten methods of DoS attack are implemented in ten different functions. Module decides which function to call based on command issued, and stops its execution once duration time expires.

*2) Killer:* `Killer` module kills processes holding ports 22, 23 and 80 and reserves these ports preventing killed applications from restarting. After that, it continually scans memory trying to find and kill similar malware created and started by other attackers.

*3) Scanner:* `Scanner` module uses telnet and random generated public IP address to check for other vulnerable IoT devices. Telnet user names and passwords are taken from table containing 62 factory default combinations [19]. If connection with random device is established successfully, IP address of vulnerable IoT device is sent to the reporting server with matching username and password.

### B. CNC server

`CNC server` is written in Google Go programming language. It first connects to MySql database using predefined credentials. Then it creates two listening sockets, one takes port 23 for telnet and other takes port 101 for API. When connection is established, initial handler decides if it is a connection from CNC registered user or a new bot registration.

TABLE I
VM Hardware Configuration

| VM | RAM (GB) | HDD (GB) |
|---|---|---|
| CNC Server | 2 | 12 |
| Bot | 2 | 12 |
| DNS Server | 2 | 12 |

TABLE II
VM Network Configuration

| VM | IP address | Subnet mask | Default gateway |
|---|---|---|---|
| CNC server | 8.8.8.1 | 255.255.255.0 | 8.8.8.1 |
| Bot | 8.8.8.2 | 255.255.255.0 | 8.8.8.1 |
| DNS server | 8.8.8.8 | 255.255.255.0 | 8.8.8.1 |

API handler first checks if API key is valid and if bot count is lesser than users maximum bot count. User command sent as a text is parsed, and byte array to be sent to bots as a command is generated. If target list does not contain whitelisted (not to be attacked) IP addresses, attack is queued. User handler generates prompt for user name and password via telnet. If user is administrator, he can use commands to add new user or check bot count beside standard attack commands. If new bot was detected, it is added to the client list. Client list's `Worker()` function ensures that bot list is up to date and distributes attack command to them. `NewAttack()` function parses user command sent via telnet. `Build()` function generates bytes to be sent to bots as a command.

### C. Loader

Loader is written in C programming language. It first creates server for downloading precompiled payloads for various architectures using `wget` or TFTP from busybox. Then it starts acting like reporting server, listening for discovered vulnerable IoT devices which can be compromised. When information about potential target is received, loader connects to it via telnet, downloads and runs payload against compromised device, thus turning it into a new bot.

## IV. Dynamic analysis

Dynamic analysis of malicious software requires creation of safe virtual environment so malicious software cannot escape created sandbox and make damage to real devices in the wild. For this purpose, Oracle VM VirtualBox was used. Three virtual machines were created with hardware and network settings given in the Table I and Table II, respectively. Kubuntu 15.10 operating system was installed on all of the machines, and they were set in internal network mode for isolation.

It can be seen that strange IP configuration was used. It's because malware relies on Google's DNS server with IP address 8.8.8.8, and it was preferred to keep as much original configuration as possible.

*1) Server configuration:* Since this software part was entirely written in Google Go programming language, it was the first thing to be installed using command:

TABLE III
USERS

| Field | Type | Key | Default |
|---|---|---|---|
| id | int(11) | PRI | NULL |
| username | varchar(15) | UNI | NULL |
| password | varchar(15) | | NULL |
| api_key | varchar(500) | | NULL |
| max_bots | int(11) | | NULL |
| admin | int(11) | | NULL |
| wrc | int(11) | | 0 |
| last_paid | timestimpe | | CURRENT_ TIMESTAMP |
| cooldown | int(11) | | NULL |
| duration_limit | int(11) | | NULL |
| intvl | int(11) | | 30 |

TABLE IV
HISTORY

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

TABLE V
WHITELIST

| Field | Type | Key | Default |
|---|---|---|---|
| prefix | varchar(15) | | NULL |
| netmask | int(11) | | NULL |

```
sudo apt-get install golang
```

MySql driver and go-shellwords were also required, and their installation required git.

```
sudo apt-get install git
go get github.com/go-sql-driver/mysql
go get github.com/mattn/go-shellwords
```

Static analysis showed that CNC server uses local database "mirai", with username "root" and password "password". It can also be noticed that database contains three tables: `users`, `history` and `whitelist`. Stricture of those tables is shown in the Table III, Table IV Table IV, respectively..

One user was inserted in created database using SQL command:

```
INSERT INTO users(username, password,
api_key, max_bots, admin, last_paid,
cooldown, duration_limit) VALUES ('test',
'test', 12345, 2, 1, UNIX_TIMESTAMP(), 1,
0);
```

After that, server still could not connect to the database, so connection string in `database.go` was changed to:
`"%s:%s@tcp(%s:3306)/%s"`
where 3306 is default port for MySql database.

*2) Bot configuration:* Build script for bot was simplified so it's compiled only for x86 architecture. It was decided to run analysis on debug version of malware, so it's worklflow could be monitored through debug messages. First run showed that there was an error somewhere in killer module. Static analysis of this module discovered that it tries to access locked value in constants table. Adding lines
`table_unlock_val(TABLE_KILLER_STATUS);`
before accessing `TABLE_KILLER_STATUS` and
`table_lock_val(TABLE_KILLER_STATUS);`
after accessing it solved this issue. Finally, to perform DoS attack in debug mode, it was necessary to disable security mechanisms that prevents attack in this mode. It's done by deleting lines

```
#ifdef DEBUG
    if(errno != 0)
        printf("errno = \%d\n", errno);
    break;
#endif
```

found in files implementing attack module.

*3) DNS server:* Malicious software uses Google's DNS server with IP address 8.8.8.8 to discover CNC server's IP address. Virtual machine with DNS server got the same IP address so malware can access it successfully. When doing analysis of malicious software, it's useful to have fake DNS server that returns fixed IP address for any query received. For this purpose, python script was used. The script is executed using command

```
sudo python recipe-491264-1.py
```

Before running this script, it's mandatory to kill dnsmasq so port 53 can be used, and to change IP address script returns to 8.8.8.8.

*4) Running analysis:* Virtual environment for Mirai analysis and reproduced steps are shown in Fig.1.

1) Bot sends DNS request for CNC server domain name
2) DNS server responses with CNC server IP address
3) Bot registers itself to the CNC server
4) Botnet user issues DDoS attack command
5) Bot starts executing DoS attack on selected target

When connected to running CNC server via telnet, user is prompted for username and password, as shown in Fig.2, first three lines. It is interesting to notice that prompt is in Russian. There were speculations that it proves that Mirai is Russian made malware or that it was implemented to confuse analysts to falsely accuse Russians.

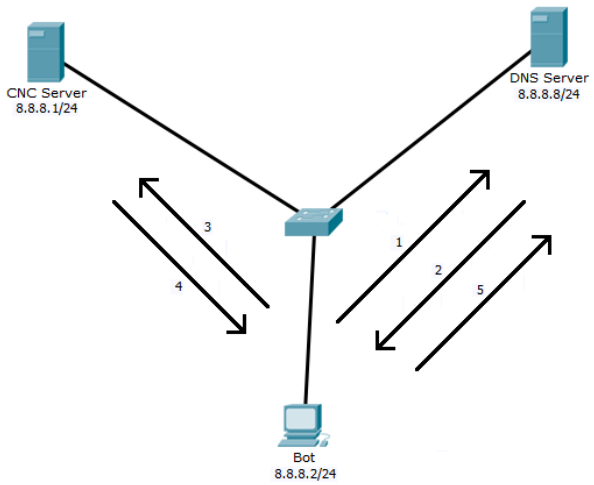


Fig. 1. Scheme

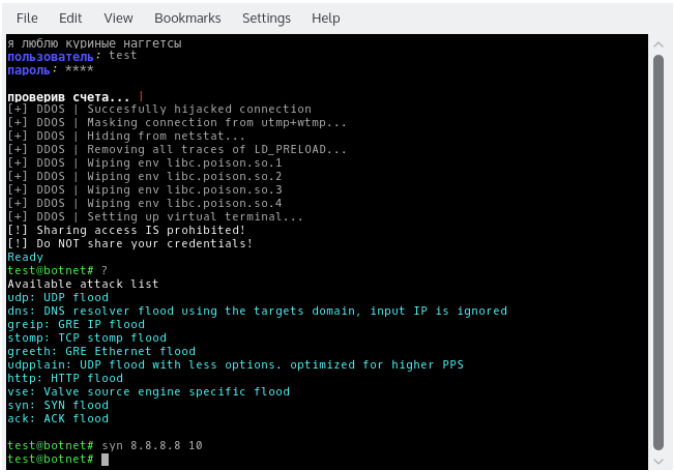


Fig. 2. Prompt

If login was successful, initial messages and virtual terminal are displayed.

Entering question mark as a command displays available attacks list, which can also be seen in Fig.2. As it can be seen, there are ten different DDoS attack methods.

1) `udp` and `udpplain` generate UDP packets with random payload and source IP address by default.
2) `syn`, `ack` and `stomp` generate TCP SYN or ACK flood.
3) `http` generates HTTP flood
4) `use` generates Valve Source Engine query flood.
5) `dns` uses slow drip DDoS attack to target authoritative DNS server.
6) `greeth` and `greip` generate GRE encapsulated Ethernet and IP packets.

It can be seen, in the source code, that there had been one more attack vector planned, but was not implemented.

```
[**] [1:10001:1] Possible TCP DoS [**]
[Priority: 0]
02/22-01:41:34.530019 8.8.8.2:9317 -> 8.8.8.8:45561
TCP TTL:64 TOS:0x0 ID:38373 IpLen:20 DgmLen:60 DF
******S* Seq: 0xB0DEDBB2  Ack: 0x0  Win: 0x0  TcpLen: 40
TCP Options (5) => MSS: 1404 SackOK TS: 88219604 0 NOP WS: 6

[**] [1:10001:1] Possible TCP DoS [**]
[Priority: 0]
02/22-01:41:44.023884 8.8.8.2:30731 -> 8.8.8.8:8826
TCP TTL:64 TOS:0x0 ID:8016 IpLen:20 DgmLen:60 DF
******S* Seq: 0x9B3B34C5  Ack: 0x0  Win: 0x0  TcpLen: 40
TCP Options (5) => MSS: 1404 SackOK TS: 88219604 0 NOP WS: 6
```

Fig. 3. TCP SYN flood detected

In order to test complete virtual setup operation, command `syn` was used to start TCP SYN flood on IP address 8.8.8.8 for 10 seconds. This command issuance can be seen as the last line on Fig.2. Generated network traffic was captured using Wireshark on DNS server which served as target for this attack.

Other types of DoS attacks generate similar network traffic.

## V. Detection

The virtual environment that we created enabled us to analyse network traffic generated by Mirai. The traffic analysis made it possible to write rules for signature-based network intrusion detection system to identify Mirai. For this purpose, Snort IDS was used. Based on analysis, it was easy to write rules for the particular type of DoS attack. For TCP SYN flood, the following rule was created:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET
any (flags: S; msg: "Possible TCP DoS";
flow: stateless; threshold: type both,
track by_dst, count 70, seconds 10; sid:
10001; rev:1;)
```

Detection message based on this rule is shown in Fig.3.

Similar rules were created for other DDoS attacks, with similar detection result.

In addition to DDoS attack detection, infection attempts could be detected. Scanner's telnet validation can be recognized in inbound or outbound network traffic. Detection rule could be written like:

```
alert tcp $EXTERNAL_NET any <> $HOME_NET
any (msg: "Possible Mirai infection";
content: "/bin/busybox MIRAI"; sid: 10003;
rev:1;)
```

To demonstrate this rule, telnet server should be installed and running, and virtual machine should have user with password like one of the defaults that Mirai uses. Mirai's scanner detection is shown in figure Fig.4

## VI. Other Observations and Mitigation Suggestions

Mirai relies on default telnet username and password to infect other devices, so the best and the easiest way to protect device from being infected is to change it's default remote access settings.

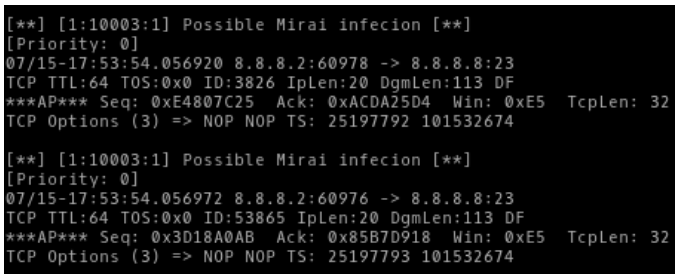


Fig. 4. Telnet scanner detected

It's relatively hard to detect using antivirus tools, since it does not leave clean signature. It implements several obfuscation techniques including sensitive strings encryption, needless calculations and debugging symbols removal which prevents attaching of gdb debugger.

It has mechanism for detection and removal of other instances of the malware on infected device. This mechanism checks predefined high port and kills process holding it. The same mechanism could be used to automatically detect and clean malware as soon as it starts running on the device.

Network exchange between bots and CNC or reporting server is not encrypted, so it is possible to create IDS rules for it's detection.

Honeypots could be used as well to log commands used for downloading and running malware, collect informations about it's usage and IP addresses involved.

Also, since Mirai stays in memory only, it could be removed from device by simply turning it off and then on again.

## VII. Conclusion

Mirai is a malware that turns infected device into bot for executing DDoS attacks. It infects IoT devices with enabled remote access via telnet and default username and password kept. Mirai is divided in three parts. CNC server provides virtual terminal for botnet users, keeps evidence of registered bots and passes attack command to them. Loader uploads and executes malware on reported vulnerable devices. Bot searches for vulnerable targets and executes DoS attack on demand. Dynamic analysis showed how bot receives command and executes DoS attack.

Based on analysis and due to its simple network behaviour it was possible to create IDS signatures for all parts of Mirai operation. That seems to be the best and easiest way to detect and stop Mirai.

Next step in research would be to create more complex virtual environment to see how bot finds vulnerable device, and how that device gets compromised.

## References

[1] B. Krebs, "Krebsonsecurity Hit with Record DDoS." https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/, 2016. [Accessed 19.5.2017.].

[2] OVH, "The DDos that didn't break the camel's VAC." https://www.ovh.com/us/news/articles/a2367.the-ddos-that-didnt-break-the-camels-vac, 2016. [Accessed 19.5.2017.].

[3] S. Hilton, "Dyn Analysis Summary Of Friday October 21 Attack." https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/, 2016. [Accessed 19.5.2017.].

[4] H. Chen and D. Wagner, "Mops: An infrastructure for examining security properties of software," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, (New York, NY, USA), pp. 235–244, ACM, 2002.

[5] H. H. Feng, J. T. Giffin, Y. Huang, S. Jha, W. Lee, and B. P. Miller, "Formalizing sensitivity in static analysis for intrusion detection," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pp. 194–208, May 2004.

[6] H. Chen, D. Dean, and D. Wagner, "Model checking one million lines of c code.," in *NDSS*, vol. 4, pp. 171–185, 2004.

[7] K. Thompson, "Reflections on trusting trust," *Commun. ACM*, vol. 27, pp. 761–763, Aug. 1984.

[8] F. B. Cohen, *A Short Course on Computer Viruses*. New York, NY, USA: John Wiley & Sons, Inc., 2nd ed., 1994.

[9] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *IEEE Security Privacy*, vol. 5, pp. 32–39, March 2007.

[10] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, pp. 6:1–6:42, Mar. 2008.

[11] Z. K. Zhang, M. C. Y. Cho, C. W. Wang, C. W. Hsu, C. K. Chen, and S. Shieh, "Iot security: Ongoing challenges and research opportunities," in *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, pp. 230–234, Nov 2014.

[12] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: Analysing the rise of iot compromises," in *Proceedings of the 9th USENIX Conference on Offensive Technologies*, WOOT'15, (Berkeley, CA, USA), pp. 9–9, USENIX Association, 2015.

[13] I. Z. Ben Herzberg, Dima Bekerman, "Breaking Down Mirai: An IoT DDoS Botnet Analysis." https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html, 2016. [Accessed 20.5.2017.].

[14] D. Web, "Investigation of linux.mirai trojan family," tech. rep., Doctor Web, 2016.

[15] E. Bertino and N. Islam, "Botnets and internet of things security," *Computer*, vol. 50, pp. 76–79, Feb 2017.

[16] K. Angrishi, "Turning internet of things(iot) into internet of vulnerabilities (iov) : Iot botnets," *CoRR*, vol. abs/1702.03681, 2017.

[17] J. v. H. Ivo van der Elzen, "Techniques for detecting compromised iot devices," tech. rep., University of Amsterdam, 2017.

[18] Anna-senpai, "Mirai-Source-Code." https://github.com/jgamblin/Mirai-Source-Code, 2016. [Accessed 19.5.2017.].

[19] Anna-senpai, "Mirai-source-code/mirai/bot/scanner.c:// set up passwords." https://github.com/jgamblin/Mirai-Source-Code/blob/master/mirai/bot/scanner.c#L124, 2016. [Accessed 20.5.2017.].