IoT honeypot: a multi-component solution for handling manual and Mirai-based attacks

Haris Šemić and Sasa Mrdovic, *Member*, IEEE

Abstract—**This paper proposes an implementation of honeypot that detects and reports telnet attacks on Internet-of-Things (IoT) devices. The honeypot operates with manual and Mirai-based attacks. A multi-component design is implemented in order to attain sufficient exposure to malicious traffic and security of collected data. Settings and additional files needed to run the honeypot are explained. Honeypot is tested using Mirai and results are discussed. After that, conclusion and directions for future work are given.**

***Keywords* - Honeypot, Internet-of-Things, manual attacks, Mirai, multi-component**

I. Introduction

Internet-of-Things (IoT) has introduced billions of special-purpose devices that connect and share their data over the Internet. The number of IoT devices on the Internet is expected to reach 24 billion by the year 2020 [1]. Those devices often have limited hardware and software capabilities which leaves little room for implementation of sufficient security mechanisms.

Username and password are usually the only security measure that IoT devices use against potential attacks and in the majority of cases passwords are not sufficiently complex and long, as described in [2]. This lack of security has left IoT devices vulnerable to various attacks that aim to take control of said devices and utilize them for various malicious purposes.

One of the most widespread tools for such purposes is Mirai malware. Mirai scans the Internet using random IP address generation with goal of finding and infecting vulnerable IoT devices, turning them into Mirai bots. After sufficient number of devices have been infected, they can collectively be used for DDoS attacks. One such attack was unleashed upon Kerbs on Security website with network traffic reaching 620 Gbs [3].

Honeypots are widely used security controls for capturing and analyzing malicious network traffic. They can emulate various web services or operating systems to induce attackers to attempt to compromise them. The main goal of honeypots is to monitor and log received data which can later be used to help prevent future similar attacks.

Honeypots were found particularly useful in analyzing malicious traffic that exploits well-known and zero-day IoT vulnerabilities. Honeypot mimics interaction between emulated device and attacker with goal of acquiring enough data for successful analysis and future attack prevention. Special effort must be put into securing collected data, lest it be tampered with by the attacker. There are various commercial honeypots available together with a large number of honeypot implementations on Github. We found out that the majority of IoT honeypot implementations deal with one type of attack or don't pay enough attention to protection of the acquired data.

In this paper we propose a multi-component low-interaction honeypot which operates with telnet traffic. Front-end components carry out interaction with attackers and are designed to be exposed and access to them is protected only by a weak, generic password. Due to various differences between coping with human attackers and coping with malware such as Mirai, two front-end components are implemented, each dealing with one out of two said types of interactions. The back-end component, protected by the firewall, receives encrypted captured data from front-end components, decrypts it, reports it to the user and stores it permanently for future access. This approach has proven successful in granting the honeypot sufficient exposure to attacks and in the same time securing captured data. We share our findings with goal of contributing towards further improvement of IoT security.

The rest of the paper is organized as follows.

Section 2 addresses the related work. Section 3 describes the implementation of our honeypot, which includes both front-end and back-end components, together with application of firewall during deployment. Analysis of captured data is described in section 4. Section 5 presents our conclusion and directions for future work.

II. Related work

With development of the Internet-of-Things (IoT), new threats are constantly emerging. Malware that would target IoT devices appeared relatively recently. In [4] authors showed increase in number of telnet-based attacks since 2014. Authors of said paper also proposed IoT honeypot which emulated IoT devices running on different CPU architectures and helped them analyze malicious telnet traffic.

In thesis [5] author implemented honeypot which deals with multiple types of attacks. Front-end module of the honeypot logs received network traffic while back-end parses and responds to queries related to any of the four protocols supported by the honeypot. We would like to point out one interesting contrast between that system and the one we propose. The attacker of that honeypot interacts solely with

Haris Šemić is with the Faculty of Electrical Engineering, University of Sarajevo, Ul. Zmaja od Bosne bb., 71000 Sarajevo, Bosnia and Herzegovina (phone: 387-62-367462; e-mail: hsemic2@etf.unsa.ba).

Corresponding Sasa Mrdovic is with the Faculty of Electrical Engineering, University of Sarajevo, Ul. Zmaja od Bosne bb., 71000 Sarajevo, Bosnia and Herzegovina (phone: 387-61-171137; e-mail: sasa.mrdovic@etf.unsa.ba).

978-1-5386-3073-0/17/$31.00 ©2017 IEEE

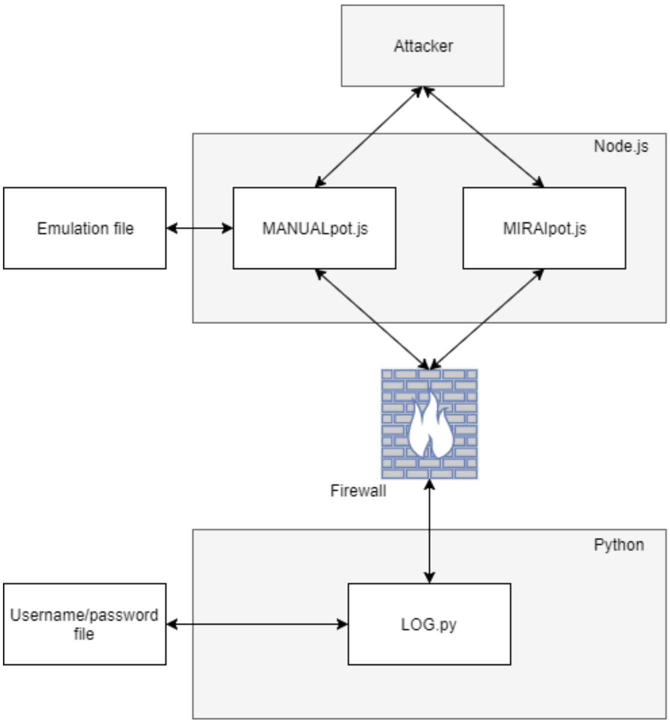


Fig. 1. Structure of the implemented honeypot

back-end component and is unaware of existence of the front-end component which logs his actions. In case of our own implementation the attacker doesn't know about the back-end component, which receives and stores log data, and interacts solely with the front-end.

We would like to mention one more paper which deals with thematic of IoT honeypots. In [6] authors implemented honeypot that uses machine learning for attaining behavioral knowledge of IoT devices. They identified this new type of honeypot as "intelligent-interaction".

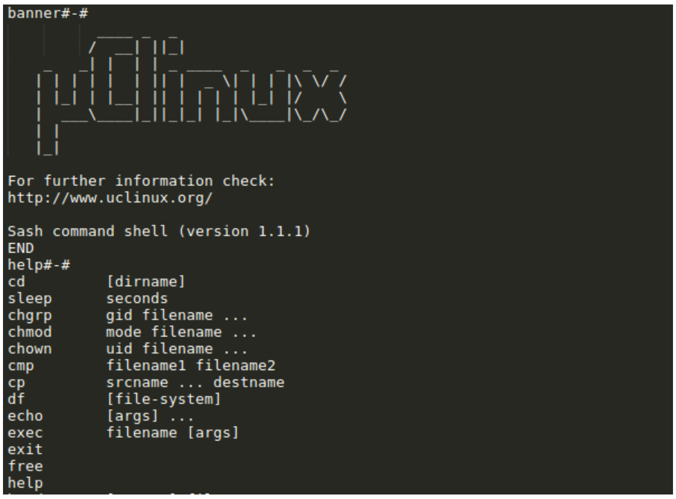
Mirai-based DDoS attacks have shown the world the danger that IoT botnets pose for global Internet security and a number of Mirai-related publications emerged. IoT botnets were analyzed in [7] including Mirai. Detailed analysis of Mirai code and behavior can be found in [8].

III. IoT Honeypot

Implemented honeypot is of low-interaction and intended mainly for research purposes. It captures manual and Mirai traffic, reports it and stores it permanently for future access. As stated before, the front-end component carries out interaction with the attacker while the back-end deals with reporting and storage. Fig. 1 shows basic structure of the honeypot.

A. Front-end

Front-end is implemented using Node.js. It doesn't offer attacker a live terminal; instead, it generates answers based on attacker's input. Due to differences in handling automatic and manual traffic front-end is divided into two separate executable scripts: `MANUALpot.js` and `MIRAIpot.js`.



```
banner#-#
      ____ _  _
     /  __| ||_|
 _  _| |   | |_ _ ____  _  _ _  _
| | | | |   | | | | _ \| | | |\ \/ /
| |_| | |__| | | | | | | |_| |/    \
|  ___\____|_||_|_| |_|\____|\_/\_/
| |
|_|

For further information check:
http://www.uclinux.org/

Sash command shell (version 1.1.1)
END
help#-#
cd          [dirname]
sleep       seconds
chgrp       gid filename ...
chmod       mode filename ...
chown       uid filename ...
cmp         filename1 filename2
cp          srcname ... destname
df          [file-system]
echo        [args] ...
exec        filename [args]
exit
free
help
```

Fig. 2. Text file for uCLinux 2.6.19

1) MANUALpot.js: The first one is `MANUALpot.js` which is tasked with dealing with manual attacks. Focus here is on what attacker sees on their terminal. There are three key aspects of communication that need to be emulated in this case:

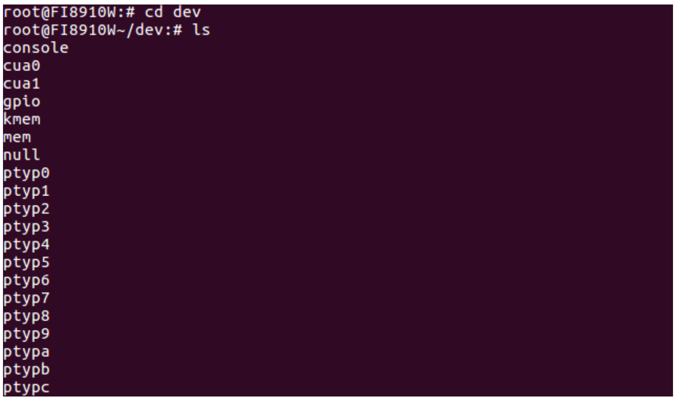
1) *Telnet login banner*: The initial message that pops up on attacker's terminal upon establishing telnet connection with the honeypot, akin to the real telnet banner expected from the emulated Internet-of-Things (IoT) device.
2) *Command responses*: Responses to various commands as implemented in the OS of the emulated device.
3) *File system*: File system of the emulated OS, focusing mainly on opening and closing directories and listing their contents.

During its execution, `MANUALpot.js` component uses complementary text file which defines aspects of emulation listed above. Every response that attacker receives based on his/her input is listed in that file. Thus, the script is executed with the following command:

```
sudo node ./MANUALpot.js path-to-file
```

If emulation of another IoT device is needed, one only needs to create dedicated text file for that device; there is no need to tamper with the code. If emulation of additional commands is needed, they are also added to the text file. Fig. 2 shows the emulation file for operating system uCLinux 2.6.19. Before creating the file said OS was run in virtual environment in order to obtain information regarding the system's behavior upon receiving inputs that would later comprise the file.

Upon establishing a telnet connection to our honeypot the attacker receives telnet banner and login and password prompts. After successful login, attacker is allowed to enter commands. At the moment of writing this paper, supported commands include: `ping`, `cd`, `ls`, `help`, `free` and `exit`. If a non-supported command is entered, appropriate response is sent. Fig. 3 shows how attacker's console looks like upon



```
root@FI8910W:# cd dev
root@FI8910W~/dev:# ls
console
cua0
cua1
gpio
kmem
mem
null
ptyp0
ptyp1
ptyp2
ptyp3
ptyp4
ptyp5
ptyp6
ptyp7
ptyp8
ptyp9
ptypa
ptypb
ptypc
```

Fig. 3. Fake directory `dev` in emulated uCLinux 2.6.19

attempting to access the directory `dev` using command `cd` and list its contents with command `ls`.

2) MIRAIpot.js: The second front-end script is `MIRAIpot.js` which handles Mirai traffic. Since the goal is to trick an automated malicious network traffic generator into perceiving our honeypot as a valid IoT device, focus here is on emulating strictly defined responses that Mirai expects during various phases of infecting an IoT device. The emulation file is not needed; all logic is defined within the code. Thus, the script is executed using the following command:

```
sudo node ./MIRAIpot.js
```

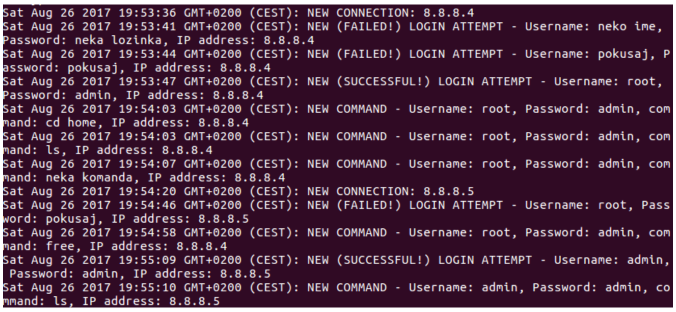
Another major difference is that Mirai requires successful telnet handshake before continuing with its attempt to infect found IoT device. The script successfully completes this phase, allowing for further data capture.

B. Back-end

Back-end is implemented using Python programming language and includes the `LOG.py` script. The main purpose of the back-end is to receive encrypted captured data from front-end components, decrypt it, transform it into readable form, report it to the user and store it permanently. We stationed it behind a firewall to ensure protection of captured data and keep attackers oblivious of its existence, but other deployment strategies are possible. Cypher used for encrypting the communication between front-end and back-end is AES-256 [9]. Fig. 4 shows how captured data is reported to the user via the `LOG.py` script.

Captured data is organized into textual files for permanent storage. One log file is created for every unique source IP address registered by our honeypot. If multiple attacks originate from the same IP address, all their data is placed within the same log file, allowing for access to entire history of attacks with one common source IP address.

`LOG.py` script uses threads to support multiple simultaneous connections to our honeypot, enabling the user to observe multiple attackers at the same time. Connections are independent, meaning that every attacker receives their own instance of terminal and is required to successfully login before entering commands. This is also shown in fig. 4 where two simultaneous connections are established towards our honeypot.



```
Sat Aug 26 2017 19:53:36 GMT+0200 (CEST): NEW CONNECTION: 8.8.8.4
Sat Aug 26 2017 19:53:41 GMT+0200 (CEST): NEW (FAILED!) LOGIN ATTEMPT - Username: neko ime,
Password: neka lozinka, IP address: 8.8.8.4
Sat Aug 26 2017 19:53:44 GMT+0200 (CEST): NEW (FAILED!) LOGIN ATTEMPT - Username: pokusaj, P
assword: pokusaj, IP address: 8.8.8.4
Sat Aug 26 2017 19:53:47 GMT+0200 (CEST): NEW (SUCCESSFUL!) LOGIN ATTEMPT - Username: root,
Password: admin, IP address: 8.8.8.4
Sat Aug 26 2017 19:54:03 GMT+0200 (CEST): NEW COMMAND - Username: root, Password: admin, com
mand: cd home, IP address: 8.8.8.4
Sat Aug 26 2017 19:54:03 GMT+0200 (CEST): NEW COMMAND - Username: root, Password: admin, com
mand: ls, IP address: 8.8.8.4
Sat Aug 26 2017 19:54:07 GMT+0200 (CEST): NEW COMMAND - Username: root, Password: admin, com
mand: neka komanda, IP address: 8.8.8.4
Sat Aug 26 2017 19:54:20 GMT+0200 (CEST): NEW CONNECTION: 8.8.8.5
Sat Aug 26 2017 19:54:46 GMT+0200 (CEST): NEW (FAILED!) LOGIN ATTEMPT - Username: root, Pass
word: pokusaj, IP address: 8.8.8.5
Sat Aug 26 2017 19:54:58 GMT+0200 (CEST): NEW COMMAND - Username: root, Password: admin, com
mand: free, IP address: 8.8.8.4
Sat Aug 26 2017 19:55:09 GMT+0200 (CEST): NEW (SUCCESSFUL!) LOGIN ATTEMPT - Username: admin,
 Password: admin, IP address: 8.8.8.5
Sat Aug 26 2017 19:55:10 GMT+0200 (CEST): NEW COMMAND - Username: admin, Password: admin, co
mmand: ls, IP address: 8.8.8.5
```

Fig. 4. Reporting attacker's actions via the `LOG.py` script

The back-end also uses complementary text file which in this case contains the list of username/password combinations used to validate login attempts. Usernames and passwords contained within the file are taken from table containing 62 factory default combinations [10]. Since both front-end scripts connect to `LOG.py`, they share the same login data. The file is automatically loaded upon executing the back-end script and is not specified upon execution which is done using the following command:

```
python LOG.py
```

IV. Analysis of captured data

We tested our honeypot using the Mirai source code [11]. Fig. 5 shows that the testing environment was comprised of four virtual machines, three of which were Mirai-dedicated: command and control server, DNS server and Mirai bot, all running Kubuntu 15.10 operating system with settings as described in [8]. Fourth virtual machine run our honeypot on Ubuntu 16.04 OS.

Fig. 6 shows communication between Mirai bot and our honeypot during Mirai's recon phase. It can be seen that, after establishing connection, doing the telnet handshake and successful login, honeypot receives several successive inputs from the Mirai bot. Four received commands after login are: `enable`, `system`, `shell` and `sh`. The purpose of these four inputs is to attain access to system's shell, if it already wasn't granted upon login.

The last received input is `/bin/busybox/ MIRAI` which aims to check validity of the targeted device. Based on response, Mirai decides if the device is valid with working telnet, in which case the device's IP address and login data are sent to command server for further infection and usage in DDoS attacks, or not, in which case the attack terminates.

It is important to point out that Mirai doesn't rely on any device-specific vulnerabilities; it relies solely on weak and factory default passwords. Simple mitigation techniques would be using passwords of sufficient length and complexity, doing

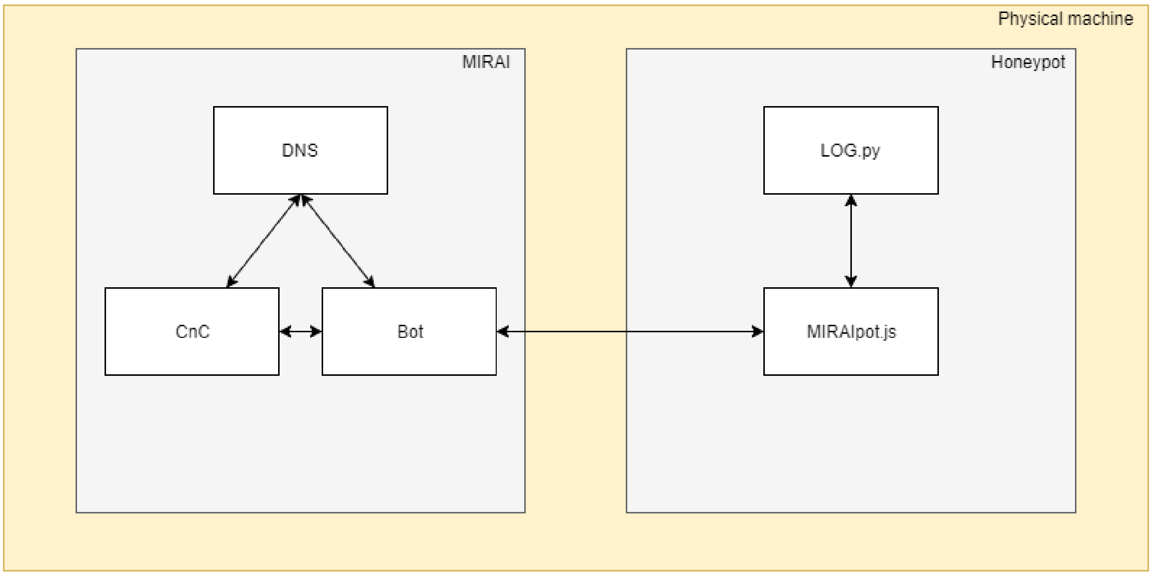


Fig. 5. Testing environment

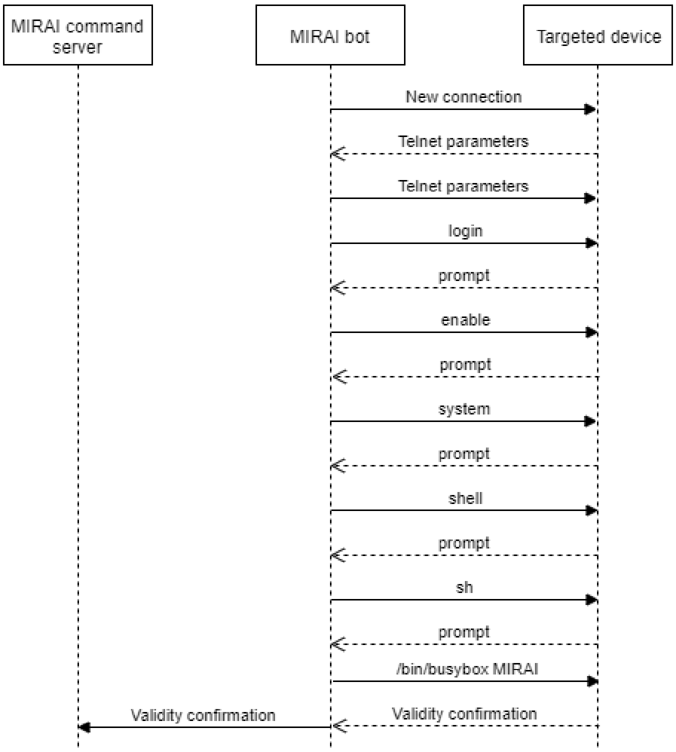


Fig. 6. Mirai recon phase

regular software and firmware updates and killing unneeded processes listening for incoming traffic.

V. Conclusion and future work

We implemented a multi-component honeypot with the front-end component that is easily detectable and penetrable by potential attackers and the back-end component that securely and permanently stores collected data. Part of our implementation was flexible design which allows easy modifications of our honeypot to emulate different IoT devices. We tested our honeypot with Mirai source code and documented our findings.

Next step in research would be inclusion of SSH module to cope with SSH-specific attacks and turning our honeypot into a honeynet that would simultaneously emulate multiple IoT devices.

References

[1] M. Sujithra and G. Padmavathi, “Internet of things–an overview,” *Avinashilingam*, 2016.

[2] D. Roe, “Top 5 Internet of Things Security Concerns.” http://www.cmswire.com/cms/internet-of-things/top-5-internet-of-things-security-concerns-026043.php, 2014. [Accessed 24.9.2017.].

[3] B. Krebs, “Krebsonsecurity Hit with Record DDoS.” https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/, 2016. [Accessed 24.9.2017.].

[4] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, “Iotpot: analysing the rise of iot compromises,” *EMU*, vol. 9, p. 1, 2015.

[5] P. Krishnaprasad, *Capturing attacks on IoT devices with a multi-purpose IoT honeypot*. PhD thesis, INDIAN INSTITUTE OF TECHNOLOGY KANPUR, 2017.

[6] T. Luo, Z. Xu, X. Jin, Y. Jia, and X. Ouyang, “Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices,” *Black Hat*, 2017.

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

[8] H. Sinanović and S. Mrdovic, “Analysis of mirai malicious software,” *25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2017.

[9] J. Daemen and V. Rijmen, “The design of rijndael: Aes-the advanced encryption standard,” *Springer Science & Business Media*, 2013.

[10] 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 26.9.2017.].

[11] Anna-senpai, “Mirai-Source-Code.” https://github.com/jgamblin/Mirai-Source-Code, 2016. [Accessed 27.9.2017.].