Journal of Information Systems Engineering & Management
2019, 4(1), em0087
ISSN: 2468-4376

Malware Threats Analysis and Mitigation Techniques for Compromised Systems

Luis Eduardo Suástegui Jaramillo [1]*

[1] *Universidad Católica Santiago de Guayaquil, Facultad de Educación Técnica para el Desarrollo, Guayaquil,* ECUADOR

***Corresponding Author:** edunashi@gmail.com

Citation: Jaramillo, L. E. S. (2019). Malware Threats Analysis and Mitigation Techniques for Compromised Systems. *Journal of Information Systems Engineering & Management*, 4(1), em0087. https://doi.org/10.29333/jisem/5742

Published: March 24, 2019

ABSTRACT

On Friday October 21, 2016 there was a Distributed Denial of Service (DDoS) attack that took place on a DNS provider. Its objective was against major websites such as GitHub, Etsy, Twitter, Netflix, and Spotify. This paper is created in order to utilize the Free and Open Source Software (FOSS) available online to identify, classify and remove malware from a compromised system. Presenting an in-depth security analysis of Mirai botnet, a malware that convert devices running Linux into remotely controlled Bots, especially IoT devices, all the compromised systems were used as part of the Mirai botnet for performing large-scale network attacks. The methods presented in this article are generic and can be used as part of an incident response strategy to mitigate any malware of the same nature.

Keywords: Linux Malware, FOSS, Incident Handling, Mirai botnet

INTRODUCTION

FOSS is open sources software that is free to be used by anyone. That is, users have the freedom to run, copy, distribute, study, change and improve the software. The benefits of using FOSS can include decreased software costs, improve security strategies and interoperability.

Computer software that serve for malicious purposes are usually called malware, from malicious software. There are variety of forms of intrusive software, trojans, rootkits, spywares, worms, etc. The most destructive is the type of malware that spreads automatically over the network from machine to machine by exploiting known or unknown vulnerabilities. That kind of malware is not only a constant threat to the integrity and confidentiality of individual computers on the internet. For example, compromised machines that an attacker can manipulate from a botnet (collection of bots in a command and control (C&C) server controlled by an attacker) are called bots. Malicious use of bots is the coordination and operation of automated attack by a botnet that can bring down almost any server through Distributed Denial of Service (DDoS) attack.

According with the National Institute of Standards and Technology (NIST), malwares are the most common external threat to most hosts, that causing widespread damage and disruption and necessitating extensive recovery efforts within most organizations. (Souppaya and Scarfone, 2013)

The process of collecting malware and analyzed it, is not easy. Nowadays, most of the malware´s example available online for educational purpose, are obtained by detailed forensic examinations of infected machines.

With the increase of new types of malwares, this can only be done for a small proportion of system comprises. In addition, sophisticated worms and viruses spread so fast in the network that the collection of they without special devices and a highly qualified staff is almost impossible. In both cases, it is necessary a very high degree of automation to handle these issues. (Provos and Holz, 2008)

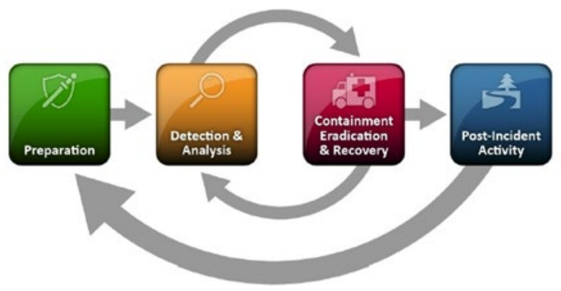


Figure 1. Incident Response Life Cycle

One of the big problems in the information security industry today, is having the ability to identify malware either as it attempts to get on the network or subsequent to the malware already being present. Several factors make this identification particularly difficult:

1) Vast amount of malware: Exists and is created on a daily basis, often results in the uselessness of intrusion detection systems (IDS) based on signature.

2) Malware integrated: Malware is often integrated in otherwise-trusted applications and sent over protocols that are traditionally allowed through firewalls and access lists.

3) Restricted amount of resources: Organizations have narrow range of resources (both human and technology) to effective control the massive amounts of traffic that pass through the network. The quantity of data traffic, both good and bad, has become so large that it is almost too much for any one organization to keep up.

4) Use of encryption: The increasing use of encryption has added another challenge for organizations trying to gain visibility into malicious traffic residing on the network.

As defined in NIST SP 800-61, Computer Security Incident Handling Guide, the incident response process has four major phases: preparation, detection and analysis, treat, and post-incident activity. Figure 1 from NIST SP 800-61 displays this incident response life cycle.

This process is going to be used in the rest of the paper as the framework.

LINUX/MIRAI BOTNET- OVERVIEW

So far, we have talked about how difficult it can become to collect and manage malware. Botnet is a network of compromised machines under the influence of malware (bot) code (Gu et al., 2008). This kind of networks can cause much harm in today's Internet. From August 4th 2016 several sysadmins were helping the cyber security community by uploading Mirai malware files and sharing their experiences through forums and specialized pages. This threat is made by a new ELF (Executable and Linkable Format) trojan which have been used to create large-scale botnets—networks of devices infected with self-propagating malware—that can execute crippling distributed denial-of-service (DDoS) attacks., the name of the binary is "mirai.*" and is having telnet attacks as main functionality to IoT devices.(US-CERT, 2016)

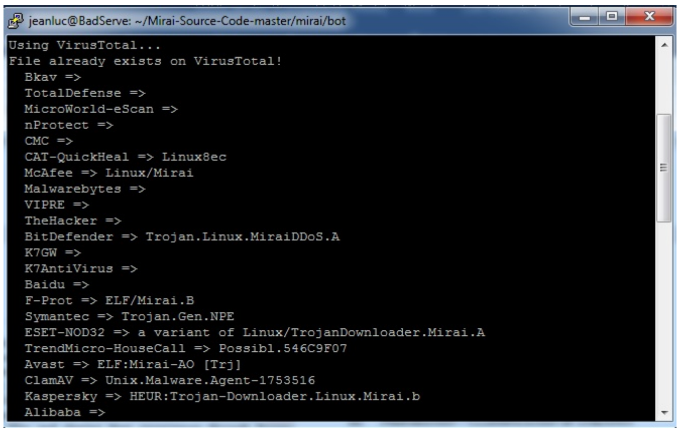
Mirai is a linux malware written in C language, the main but not exclusive objective of this malware is the infection of routers and IP cameras by continuously scanning ports 22, 23, 5747, etc. Uses a brute force technique for guessing passwords through a dictionary attacks. Once connected to an IoT, attempts to login, gain access, and infect the device. The infected device then scans other networks looking for more IoT devices and launches DDoS attacks (DAC, 2012). The botnet control structure usually uses HTTP or IRC for the remote command channel, sometimes proprietary communication protocols can be used, even social networks like twitter can be employed to perform control over a bot. To avoid a single point of failure some botmasters (bot administrator, could be a person or system) use peer-to-peer(P2P) communication mechanisms. A command and control (C&C) server lets an attacker manipulate infected machines. Usually, two types of commands are implemented in the server, updates and DDoS attacks. DDoS attacks include SYN and UDP flooding (Provos and Holz, 2008).

According with Imperva Incapsula security team investigation since the source code was published, 49.657 unique IPs hosted Mirai-infected devices. These were primarily surveillance systems and routers with default settings. On the whole, IP addresses of Mirai-infected devices were spotted in 164 countries (Herzberg et al., 2016).

Table 1 from Imperva's DDoS attacks report, shows the top countries of origin of Mirai DDoS attacks.

Table 1. Top countries of origin of Mirai DDoS attacks

Country	*% of Mirai botnet IPs*
Vietnam	12.8%
Brazil	11.8%
United States	10.9%
China	8.8%
Mexico	8.4%
South Korea	6.2%
Taiwan	4.9%



```
Using VirusTotal...
File already exists on VirusTotal!
  Bkav =>
  TotalDefense =>
  MicroWorld-eScan =>
  nProtect =>
  CMC =>
  CAT-QuickHeal => Linux8ec
  McAfee => Linux/Mirai
  Malwarebytes =>
  VIPRE =>
  TheHacker =>
  BitDefender => Trojan.Linux.MiraiDDoS.A
  K7GW =>
  K7AntiVirus =>
  Baidu =>
  F-Prot => ELF/Mirai.B
  Symantec => Trojan.Gen.NPE
  ESET-NOD32 => a variant of Linux/TrojanDownloader.Mirai.A
  TrendMicro-HouseCall => Possibl.546C9F07
  Avast => ELF:Mirai-AO [Trj]
  ClamAV => Unix.Malware.Agent-1753516
  Kaspersky => HEUR:Trojan-Downloader.Linux.Mirai.b
  Alibaba =>
```

Figure 2. Output generated by avsubmit.py

Table 2. Mirai shellcode example

Offset	*Instruction*	*Byte codes*
1f	mov %rax,%rsi	48 89 c6
22	mov $0x1,%edi	bf 01 00 00 00
27	Callq 6ab	e8 6b 06 00 00

At this time Mirai malware is having a low antivirus detection ratio, even in the architecture of x86.

With the free software avsubmit.py create by Michael Ligh is possible to analyses a file using the VirusTotal engine from the command line. Figure 2 shows part of the output generated by the program.

From the 58-online antivirus that VirusTotal use to analyze a file, just 14 of then identify the malware as a possible threat.

PREPARATION – CREATING A GAME PLAN

One of the most common tasks malware analysts perform is classification of content. This content can be something well known or completely new. The classification process starts with the simple, as detecting the file type and then move to the more complex, as detecting similarities with other files of the same nature and determining patterns of comportment.

ClamAV is a cross-platform open source (GPL) anti-virus engine owned by Sourcefire. It provides a fast and flexible framework for detecting malicious code and artifacts (DAC, 2012). For our purpose, we are going to use ClamAV to establish a baseline to confirm or deny a false positive of Mirai malware or any kind of malware that is based on this code. To do that we are going to create a custom binary signature based on the source code of the original malware and create our own data base. For example, consider the following disassembly of shellcode from Mirai malware Table 2.3.

Is possible to use the byte code value to create a binary signature (1). Staring with ClamAV version 0.96, the basic signature format is deprecated in favor of an extended signature format. We are going to use the extended signature format (2), which consist of the following four fields separated by colons.

Any_name:0:*: 4889c6bf01000000e86b060000 (1)

SigName;Target;Offset;Byte_Codes (2)

The SigName is a unique descriptive name for the signature, target parameter can be any value from 0 to 9 where each number has a different propose. For example, 0 is for any file type and 9 for Mach-O binaries (Database, Ok, and Clamav, 2007).

Table 3. Yara rule syntaxes

```
rule silent_banker : banker{
meta:
description = “This is just an example”
thread_level = 3
in_the_wild = true }

strings:
$a = {6A 40 68 00 30 00 00 6A 14 8D 91}
$b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
condition:
$a or $b
}
```

DETECTION & ANALYSIS- CLASSIFICATION OF UNKNOWN

Once we have our costume database is time to take the next step and classify our malware. Yara is an open-source and multi-platform tool (it works with most hosts running Windows, Linux, or Mac operating systems) aimed at (but not limited to) helping malware researchers to identify and classify malware samples (“Yara 3.6.0 documentation”, 2017). Using Yara we can create rules that detect strings, instruction sequences, regular expressions, byte patterns, and so on. Today there are various IDS, IPS (intrusion prevention system) and antivirus solutions that use Yara rules to detect or prevent malwares, its popularity comes from its simple and efficient way of writing rules. Table 3 shows a simple example of how a Yara rule syntaxes looks like.

Packers is a code obfuscation technique/software used by software developers for legitimate purposes, some of which include file reduction or as an attempt to stop illegal software copying. Sometime malware developers use packers to evade antivirus or tools like Yara. Protecting it with technique/software like packers to prevent anybody from viewing the malware’s code until it is placed in memory. While this is true, the number of malware than can be detected using well done signatures and Yara’s engine are considerably bigger than those malware that because their nature can evade this tool. The National Cybersecurity and Communications Integration Center from USA use custom Yara’s signatures to disseminate the intelligence needed by asset owners to defend their networks. The signature files support the documentation of both byte-sequences and string matches that occur in the malware, as well as logic operators that support very robust and precise conditions to reduce the incidence of receiving false positives (National Cybersecurity and Communications Integration Center, 2015).

Converting ClamAV signature to Yara format is possible and can be done in an automatic way, with a script written in python by Matthew Richard, changing from our costume database to and signature that can be understand by Yara. This approach work for certain types of malware but for those who change over time or autodelete, another way of treatment it is necessary and demand a little more research work. First at all is necessary understand the malware, know what exactly it does and if a modification is made (most of the time they need to do). Then knowing what file was modified create a hash for that file before it has been altered and then create a Yara signature base in that modification. For example, during the execution of Mirai, the malware will try to open the /dev/watchdog or /etc/watchdog file (depends where is watchdog) in read-write state to disable it. Watchdog is a demon or a piece of hardware that restart the system without needing any administrator intervention. Also, if any other malware of the same nature is in the system like Zollard or Qbot, Mirai will kill that process. Then it will hide his process name and open the UDP/53 port to access Google DNS server in 8.8.8.8 and established a connection. The malware will detect the outbound interface and opens a random TCP/port to the C&C (command and control server) IP address. At this point is performing several decoding for strings, which will be resulted in the targeted malware. The /dev/watchdog execution make sure the malicious opened backdoor port is up and used because if the system restarts for any reason the entire process will not work. Upon execution, the malware will be self-deleted to avoid the trace, but the process is still running(“MalwareMustDie! - Malware Research Group”, 2016).

Knowing that certain modifications were made to watchdog demon, it is already possible to create our new rule. Table 4 shows the results of the previous research about the malware’s behavior.

Table 4. Custom Yara rule for Mirai malware

```
rule Mirai_Botnet {meta: hash2 =
“05c78c3052b390435e53a87e3d31e9fb17f7c76b00df2814313bca24735ce81c” hash3
= “20683007a5fec1237fc09224af40be029b9548c62c693844624089af568c89d4”
strings:
$x2 = “/dev/misc/watchdog” fullword ascii
$x3 = “/dev/watchdog” ascii
condition:
uint16(0) == 0x457f and filesize < 175KB and ( ( 1 of ($x*) and 1 of ($s*) ) or 4 of
($s*) ) }
```

Table 5. Honeyd configuration

#Cisco C870 router (IOS 12.4)	#Apple TV (iOS 4.3)
create routerone	create appletv
set routerone personality “Cisco C870 router (IOS 12.4)”	set appletv personality “Apple TV (iOS 4.3)”
set routerone ethernet “Cisco Systems”	set appletv ethernet “Apple Computer”
set routerone default tcp action reset	set appletv default tcp action reset
add routerone tcp port 23 “scripts/embedded/router-telnet.py”	add appletv tcp port 161 “scripts/embedded/snmp/fake-snmp.pl”
dhcp routerone on eth0	dhcp appletv on eth0

CONTAINMENT ERADICATION & RECOVERY

Eradicate Mirai malware is simple, is necessary just disconnect the device from the network and perform a reboot. Mirai malware exists in dynamic memory, rebooting the device clears the malware (US-CERT, 2016). Then is necessary to change the default password with something good like as defined in NIST 800-118. At this point is important remember that all of this happened because the IoT device was not configure in a proper way leaving the default password. Mirai takes advantage of this vulnerability and through a dictionary attack against its objective gain system access. Only to emphasize the importance of device hardening, performing a quick investigation with shodan’s search engine, with “default password” as the parameter revealed that 65,983 devices around the world were left by their default values (Suastegui, 2018).

Because the purpose of this paper is to mitigate any malware of any nature, it is important to say that from Nessus 6.8, file system scanning functionality based on Yara signatures was introduced that could look for specific file hashes of files on disk (Tenable, 2016). Although Nessus is not a FOSS tool it has a community edition which has no price. Is a good alternative to perform multiple device scanning with a custom signature without the need to install Yara. Once the malware is located on a device, using ClamAV we perform the elimination. Using YARA rules with ClamAV 0.99 is simple - just is necessary to place YARA rule files into the ClamAV virus database location and run the antivirus.

POST INCIDENT ACTIVITY- FINAL STEP

Learning and improving is one of the most omitted tasks. It can make the different between a successful incident treatment or a vulnerability that remains persistent. As part of the ISMS (information security management system) after developing our risk treatment plan, for us the Yara custom signature, it is extremely necessary to monitor and improve the network looking for anomalies because, the threat may have changed its behavior and therefore the signature must be changed too (ISO, 2011). A Raspberry Pi with Honeyd installed can do this monitoring.

Honeyd is a low-Interaction honeypot framework that instrument thousands of Internet addresses with virtual honeypots and corresponding network services. Supports the IP protocol suites (Provos and Holz, 2008) and responds to network requests for its virtual honeypot according to the services that are configured for each virtual honeypot. It is available as an open source software released under the GNU Public License. Because a honeypot has no production value, any attempt to contact it is suspicious and saved in a database. That is why; forensic analysis of data collected from honeypots is less likely to lead to false positives than data collected by networks detection systems (NIDS). Honeypots can run any operation system and any number of service (SANS, 2016). The configured services determine the vectors available to an adversary for compromising or probing the system. Table 5 shows a simple example of honeyd configuration.

In this example, two different devices are emulated. The interesting part is that one of each has different ports enable with a script that emulate a service. For example, Cisco C870 has a port emulation that from the point of view of a malware is port 23 (Telnet) and is enable. The raspberry is waiting for someone try to connect, if a request is performed is a symptom that a new malware is on the network and attempts to escalate privileges. At this point is necessary capture the malware with a high-interaction honeypot like T-Pot (DTAG, 2015) or HoneyDrive (BruteForce, 2012) and then performance all the incident life cycle one more time.

CONCLUSIONS

In this paper, we have presented the NIST incident Response Life Cycle and how with open source software identify, classify and remove malware in a compromised system. The cornerstone of this approach is the versatility that FOSS tools can give us to mitigate new types of malware attacks that, even modern anti-malware solutions available today can't solve. This is because every day many new possible types of malware are discover and many antivirus companies are simply unable to cope with the demand. Resulting in sending security updates to users once they are already infected, sometimes even it can take days before this happens. We evaluate some FOSS tools in order to mitigate Mirai malware, but this approach will also work to remove any other type of threat of the same nature not contemplated in this paper. Finally, the main message of this paper is that is not necessary to have expensive computer security equipment to identify, classify and remove malware from a compromised system.

ACKNOWLEDGEMENTS

I would like to show my gratitude to Rayner Durango and Vladimir Sanchez Padilla for sharing their pearls of wisdom with me during the course of this paper.

REFERENCES

BruteForce. (2012). *HoneyDrive honeypot bundle distro*. Available at: https://bruteforcelab.com

DAC, U. (2012). *Clam AntiVirus User manual*, 0–57. https://doi.org/10.1007/SpringerReference_28001

Database, C. V., Ok, C. V. And Clamav, T. (2007). *Creating signatures for ClamAV*, 1–20.

DTAG. (2015). *T-Pot: A Multi-Honeypot Platform*. Available at: https://dtag-dev-sec.github.io/

Gu, G., Perdisci, R., Zhang, J. and Lee, W. (2008). *BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection*. Available at: http://static.usenix.org/events/sec08/tech/full_papers/gu/gu_html/

Herzberg, B., Bekerman, D. and Zeifman, I. (2016). *Breaking Down Mirai: An IoT DDoS Botnet Analysis*. Available at: https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html

ISO. (2011). ISO 27005:2011 - Information security risk management. *Iso 27005:2011*, *2011*.

MalwareMustDie! - Malware Research Group. (2016).

National Cybersecurity and Communications Integration Center. (2015). *Using yara for malware detection, (June)*, 2015.

Provos, N. and Holz, T. (2008). *Virtual Honeypots*. Boston: Pearson Education.

SANS. (2016). *SANS - Information Security Resources - Information*. Available at: https://www.sans.org/security-resources/idfaq/what-is-a-honeypot/1/9

Souppaya, M. and Scarfone, K. (2013). *Guide to Malware Incident Prevention and Handling for Desktops and Laptops*. https://doi.org/10.6028/NIST.SP.800-83r1

Suastegui, L. (2018). Detecting malware capabilities with FOSS: lessons learned through a real-life incident. In *Information Systems and Technologies (CISTI), 2018 13th Iberian Conference on* (pp. 1-5). IEEE.

Tenable. (2016). Threat Hunting with YARA and Nessus. Available at: https://www.tenable.com/blog/threat-hunting-with-yara-and-nessus

US-CERT. (2016). *Heightened DDoS Threat Posed by Mirai and Other Botnets*.

Yara 3.6.0 documentation. (2017). Available at: https://yara.readthedocs.io/en/v3.6.0/