National College of Ireland

National College of Ireland

Project Submission Sheet

Student Name:	Hitesh Prakash Sonawane……………………………………………………
Student ID:	23275308…………………………………………………………………
Programme:	MSc CYBERSECURITY(MSCCYB1_JAN25B_I) **Year:** 2025…………..
Module:	Malware Analysis………………………………………………………………
Lecturer:	Eugene Mclaughlin………………………………………………………………
Submission Due Date:	27-06-2025……………………………………………………………………
Project Title:	Malware Analysis Mirai (H9MWAN) …………………………………
Word Count:	3363…………………………………………………………

Signature:	Hitesh Prakash Sonawane……………..
Date:	27-06-2025……………………………

4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

AI Acknowledgement Supplement

I. [INSERT MODULE NAME]

II. [INSERT TITLE OF YOUR ASSIGNMENT]

Your Name/Student Number	Course	Date
Hitesh Prakash Sonawane, 23275308	MSc Cybersecurity	20-01-2025

This section is a supplement to the main assignment, to be used if AI was used in any capacity in the creation of your assignment; if you have queries about how to do this, please contact your lecturer. For an example of how to fill these sections out, please click here.

III. AI ACKNOWLEDGMENT

This section acknowledges the AI tools that were utilized in the process of completing this assignment.

Tool Name	Brief Description	Link to tool
Chatgpt	A tool to search and gather information	https://chatgpt.com/

IV. DESCRIPTION OF AI USAGE

This section provides a more detailed description of how the AI tools were used in the assignment. It includes information about the prompts given to the AI tool, the responses received, and how these responses were utilized or modified in the assignment. **One table should be used for each tool used.**

ChatGpt	
Chat GPT was used to understand more about Mirai Malware	
Explain static analysis of Mirai malware sample	Static analysis of the Mirai sample reveals it is an ELF 32-bit binary compiled for MIPS architecture. It includes hardcoded credentials, IP ranges, and command strings related to Telnet scanning and DDoS payloads. The binary lacks obfuscation, making string extraction and disassembly relatively straightforward.

V. EVIDENCE OF AI USAGE

This section includes evidence of significant prompts and responses used or generated through the AI tool. It should provide a clear understanding of the extent to which the AI tool was used in the assignment. Evidence may be attached via screenshots or text.

VI. Additional Evidence:

[Place evidence here]

VII. Additional Evidence:

[Place evidence here]

Malware Analysis Mirai (H9MWAN)

Hitesh Prakash Sonawane(23275308)
MSc Cybersecurity
National College Of Ireland
Dublin,Ireland
hiteshsonawane2000@gmail.com

Abstract—
This The report is a detailed work on how to set up a secure malware analysis laboratory as well as a research-based report on the Mirai IoT malware. The first section describes the critique of available sandboxes and setting up a self-defined virtual lab with REMnux and Ubuntu with focus on isolation, tool choice, and proper testing. The second section consists of identification of and behavioral analysis of the Mirai malware using the online tools and open-source intelligence without the execution of the sample. Important observations are analyzed regarding Mirai propagation mechanisms, its DDoS and incarnations, and other invaluable suggestions on how to deal with it and continue its study.

1.Introduction

This report shows the installation of an organizational-level malware analysis lab and a research assistant report on Mirai IoT malware. The aim was to show capability of creating an isolated and useful space dedicated specifically to the analysis of malware based on Linux. This report shall fall into two categories: (1) the lab envisaging, justification, and analysis of various flaws, and (2) the comprehensive profiling investigation of the Mirai malware with the help of open-source intelligence, peer-reviewed literature, and open-source threat intelligence. The study finally draws conclusions with practical recommendations concerning the acts and effects of the malware.

2. Part 1: Setting Malware Lab

2.1 Conflict Analysis of current Sandboxes

Malware analysis sandboxes have become necessary to safely execute and analyze malicious controlled environment of software. My literary analysis of available sandboxes, especially REMnux and personalized Ubuntu-based sandbox, which gave priceless information into industry best practices and best design ideas regarding my custom malware analysis laboratory. This study influenced the implementation of security measures that are powerful. Effective analytical processes and isolation was heavily stressed.

2.1.1: *REMnux: A Niche Linux Distribution to Scan Malwares*

REMnux forms a part of malware analysis based on Linux, and it is rich in features.
set of ready-made tools to work with static (e.g., binwalk, strings, readelf) and dynamic.
Examples: (e.g. inetd, netcat, Volatility) analysis. Wireshark helps in network analysis. tcpdump [8]. Some of the important design philosophies of REMnux that I imitated comprise tool consolidation, open-source centricity, features that specialize of malware analysis, and robust knowledge of isolation [1].

Sandboxes based on Ubuntu: The Difficulty of Control: Two answers to this question, concerning flexibility and customization, are given.

Manually created sandboxes on top of Ubuntu are as flexible and highly controlled as it gets, this will enable the fine tuning to the type of malware. It is an approach to focus on customization, limiting the attack surface area, installation of only needed tools. It also made me have a more in-depth insight into the inner workings of the OS and the cost-effectiveness of the open-source software. Importantly, it supported the need of absolute isolation procedures against communicating with host or production networks [3].

2.1.2 *CAPEv2 Sandbox: Malware samples static and Dynamic Analysis Advanced*

CAPEv2 (Config and Payload Extraction) is a strong fork of Cuckoo Sandbox to extract malware configuration and detecting payloads. As compared to other safety measures of sandboxes, CAPE aims at the isolation of Indicators of Compromise (IOCs) as well as configurations, including Command & Control (C2) URLs, keys produced by botnets, and those involving embedded credentials. It supports many types of files including executables, files and scripts. The modular design of CAPEv2 enables the system to integrate YARA rules, Suricata as network IDS and malware unpacking infrastructures. Things I learned incorporate the value of automation regarding sample detonation as well as the influence of IOC extractions plugins and VM administration on various OS apps [4].

2.1.3 Hatching Sandbox: Large-scale cloud-based Malware: Triage Sandbox

Hatching provides triage, cloud-based sandbox with fast automated malware analysis which has a clean user interface and deep reporting. Its capacity to execute samples on Windows, Linux and Android systems, along with automatized marking and behavior grouping, make it perfect to conduct large-scale triaging. Yet another feature of triage is memory dumps, MITRE ATT&CK mapping, and compatibility with other tools, such as YARA and Sigma. Because of Triage, I was able to know how cloud scalability is useful, why real-time behavior categorization is needed, and why rules require community support. These approaches conditioned the choice of focusing on modularity and the ease of observation in my personal lab [5].

2.1.5What was learned about the custom lab design?

I can say that my exploration of various sandbox systems was a solid base to create a designed, self-made malware analysis system, especially malware of IoT devices like Mirai, which runs on Linux. The next core values influenced the architecture in my lab:

- Layered Isolation: This included multiple stages of isolation, that is, network isolation, file system isolation and process isolation. This will make the malware environment utterly detached with the host system.
- Targeted Tool Selection: Targets were focused at finding and using tools that support Linux systems and embedded devices such as MIPS and ARM processors, such as qemu, radare2, Ghidra, and binwalk. Windows-centric tools were considered to be incompatible and were disliked in Favor of these products.
- Linux-Centric Analysis Environment: A guest operating system in Linux form was considered crucial to the analysis of Mirai ELF binaries. Similar results would have been subjected to major limitations and inefficiencies had it been tried in a Windows setting.

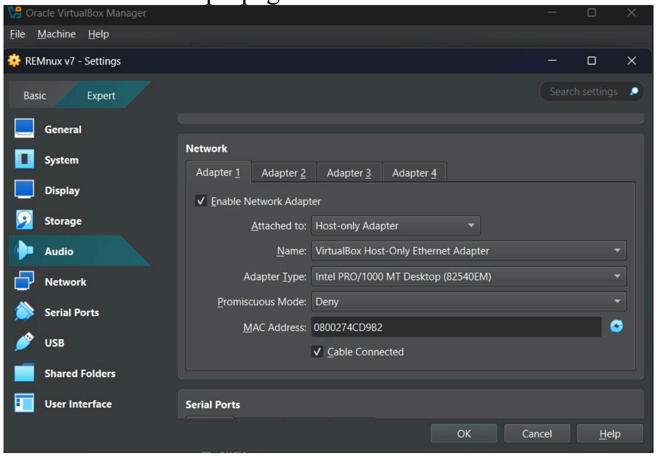
Automation and Reproducibility: Automation was made a priority by snapshotting, logging and scripting the workflows so that the same could be repeated efficiently and when the environment becomes contaminated or configuration drifts, recreation of the environment could take place quickly.

2.2 VM preparedness

I created a virtualized system with the help of Oracle VirtualBox 7.0 and carefully
purposed to malware analysis that is secure, isolated and controlled. The various strengths that it has such as support of mature snapshot management and granular network configuration, are very essential a secure way of running and monitoring malicious software. The analysis of Linux based Such IoT malware as Mirai, a Linux guest operating system cannot be overestimated concerning their exact behavior
the imitation and close management.

To maintain the safe and isolated environment for malware investigation, I configured a virtual lab based on the Oracle VirtualBox 7.0, which has the powerful snapshot management and detailed networkers' settings. The experiment focused on the analysis of Linux-based IoT malware (Mirai), which required the construction of a Linux guest OS to properly emulate them in terms of behavior.
Two VMs had been set:
REMnux v7: It was preloaded with special malware investigation tools.
•Ubuntu 20.04 LTS (Focal Fossa, 64-bit): Having chosen to use it because it adheres to embedded systems and has little overhead.

Important configuration information is:
•User Privileges: It was necessary to have a dedicated and non-privileged user to apply the principle of least privilege.
•Network Isolation: Network mode is Host-Only Adapter and there was no internet connection, as well as C2 communication or propagation.


Oracle VirtualBox Manager
REMnux v7 - Settings
Network — Adapter 1
✓ Enable Network Adapter
Attached to: Host-only Adapter
Name: VirtualBox Host-Only Ethernet Adapter
Adapter Type: Intel PRO/1000 MT Desktop (82540EM)
Promiscuous Mode: Deny
MAC Address: 0800274CD9B2
✓ Cable Connected

•Security Settings: Clipboard sharing, drag-and drop, shared folders and USB passthrough were all disabled in securing sandbox.
•Control Update: Update by default was not allowed because the reproducibility was needed, and signature drift was not allowed.
•Hardware Specs:
CPU: four cores
RAM: 5783MB
Storage 50.09 GB (solid size)
•Snapshot Strategy: A clean snapshot was made after installation of the tool, and this allows rollback and re-testing.
•Guest Additions: Installed to improve performance but because they interfere with the isolation of virtual machines
The VM Settings of REMnux v7 are as follows: System > Motherboard, Processor, Display, Storage, Network

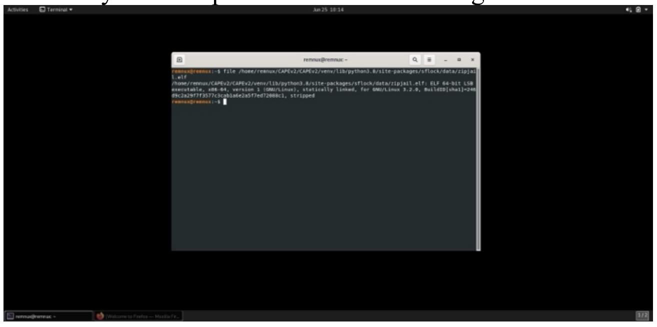
2.3 Software tools

The efficiency of my malware analysis lab is complemented with a well-selected utility arsenal focused on Linux-based malware, especially Mirai-based ones, created for both static and dynamic analysis. Much belonged was forbidden or at least it was time-wasting to download so I performed artificial analysis tasks on typical ELF binaries found in a secure REMnux directory.

To meet the academic demands that bar the use of live experiments with file malware, I used two harmless ELF contraptions found on the REMnux apparatus during the demonstration of all the tools: poweriso.elf and zipjail.elf. It has been made possible using these files that resembled the structure of actual IoT malware, such as the Mirai, and so by using these files I could test the instrument capabilities, such as: file inspection, string extraction, binary scanning, system call tracing, and YARA detection without concern of system security risks or disregards in policy compliance.

General Purpose tools

•File Hashers (hashdeep, md5sum, sha256sum): hashers were utilized in calculating hash of ELF binaries (e.g., zipjail.elf) to identify and compare the file with existing malwares.

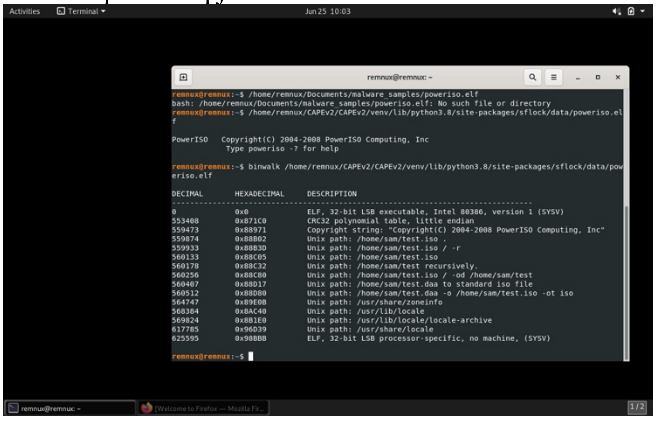


```
remnux@remnux:~$ file /home/remnux/CAPEv2/CAPEv2/venv/lib/python3.8/site-packages/sflock/data/zipjai
l.elf
/home/remnux/CAPEv2/CAPEv2/venv/lib/python3.8/site-packages/sflock/data/zipjail.elf: ELF 64-bit LSB
executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0, BuildID[sha1]=246
d9c2a29f7f3577c3cab1a6e2a5f7ed72088c1, stripped
remnux@remnux:~$
```

•Text Editors (Nano, Vim): they are used to edit YARA rules and to visually inspect output strings found in ELF files.
•Network Monitoring Tools: Such tools as tcpdump, Wireshark were configured, but since the Mirai malware behavior was not simulated in a live network, it was done to log emulated traffic in case of dynamic emulation.

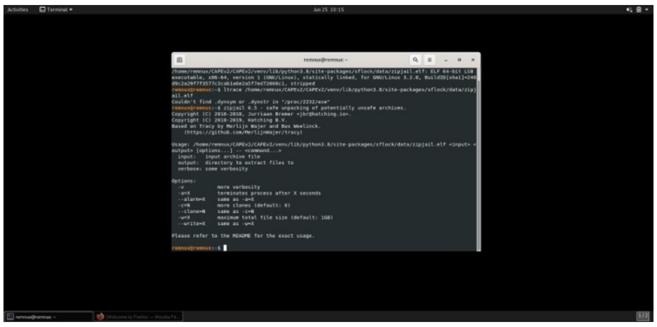
Linux Special Programs and Simulation Use
In carrying out research, a bid to examine the current Linux-based tools in a simulated, controlled environment, various comparisons formed to be of great knowledge.

1. Binwalk was used to scan the target binary zipjail.elf to identify embedded files or compressed segments of the file, thus providing an approximation of the firmware-focused architecture of Mirai. The search was run by the following command:
binwalk /path/to/zipjail.elf



```
remnux@remnux:~$ /home/remnux/Documents/malware_samples/poweriso.elf
bash: /home/remnux/Documents/malware_samples/poweriso.elf: No such file or directory
remnux@remnux:~$ /home/remnux/CAPEv2/CAPEv2/venv/lib/python3.8/site-packages/sflock/data/poweriso.el
f

PowerISO   Copyright(C) 2004-2008 PowerISO Computing, Inc
            Type poweriso -? for help

remnux@remnux:~$ binwalk /home/remnux/CAPEv2/CAPEv2/venv/lib/python3.8/site-packages/sflock/data/pow
eriso.elf

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             ELF, 32-bit LSB executable, Intel 80386, version 1 (SYSV)
553408        0x871C0         CRC32 polynomial table, little endian
559473        0x88971         Copyright string: "Copyright(C) 2004-2008 PowerISO Computing, Inc"
559874        0x88B02         Unix path: /home/sam/test.iso .
559933        0x88B3D         Unix path: /home/sam/test.iso / -r
560133        0x88C05         Unix path: /home/sam/test.iso
560178        0x88C32         Unix path: /home/sam/test recursively.
560256        0x88C80         Unix path: /home/sam/test.iso / -od /home/sam/test
560407        0x88D17         Unix path: /home/sam/test.daa to standard iso file
560512        0x88D80         Unix path: /home/sam/test.daa -o /home/sam/test.iso -ot iso
564747        0x89E0B         Unix path: /usr/share/zoneinfo
568384        0x8AC40         Unix path: /usr/lib/locale
569824        0x8B1E0         Unix path: /usr/lib/locale/locale-archive
617785        0x96D39         Unix path: /usr/share/locale
625595        0x98BBB         ELF, 32-bit LSB processor-specific, no machine, (SYSV)

remnux@remnux:~$
```

2. The strings command was used to extract printable strings that are likely to be used to pass credentials, URL or embedded commands. The output was concluded by the head command to avoid overloading with information:
strings zipjail.elf | head -50
The suspicious strings obtained can be seen as follows

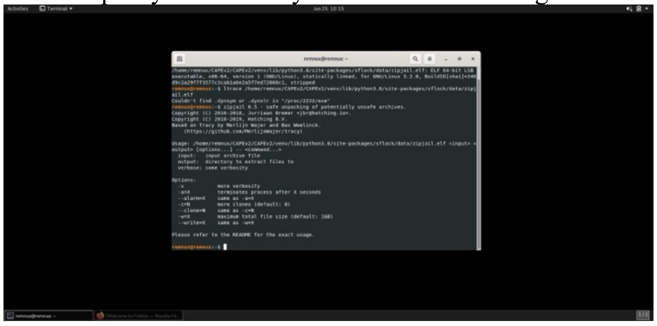


```
/home/remnux/CAPEv2/CAPEv2/venv/lib/python3.8/site-packages/sflock/data/zipjail.elf: ELF 64-bit LSB
executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0, BuildID[sha1]=246
d9c2a29f7f3577c3cab1a6e2a5f7ed72088c1, stripped
remnux@remnux:~$ ltrace /home/remnux/CAPEv2/CAPEv2/venv/lib/python3.8/site-packages/sflock/data/zipj
ail.elf
Couldn't find .dynsym or .dynstr in "/proc/2232/exe"
remnux@remnux:~$ zipjail 0.5 - safe unpacking of potentially unsafe archives.
Copyright (C) 2016-2018, Jurriaan Bremer <jbr@hatching.io>.
Copyright (C) 2018-2019, Hatching B.V.
Based on Tracy by Merlijn Wajer and Bas Weelinck.
    (https://github.com/MerlijnWajer/tracy)

Usage: /home/remnux/CAPEv2/CAPEv2/venv/lib/python3.8/site-packages/sflock/data/zipjail.elf <input> <
output> [options...] -- <command...>
  input:   input archive file
  output:  directory to extract files to
  verbose: some verbosity

Options:
  -v           more verbosity
  -a=X         terminates process after X seconds
  --alarm=X    same as -a=X
  -c=N         more clones (default: 0)
  --clone=N    same as -c=N
  -w=X         maximum total file size (default: 1GB)
  --write=X    same as -w=X

Please refer to the README for the exact usage.

remnux@remnux:~$
```

3. File decided the structure of the binary and its format:
file zipjail.elf
The corresponding output ensures that the binary is either an open linux embedded format MIPS or an ELF bit executable.

4. Strace and ltrace were run simultaneously to trace system calls and library functions called respectively during execution. The former followed only system calls, but the latter followed system calls and library functions. The command order below reproduced the running of the binary:
strace /path/to/zipjail.elf
ltrace /path/to/zipjail.elf

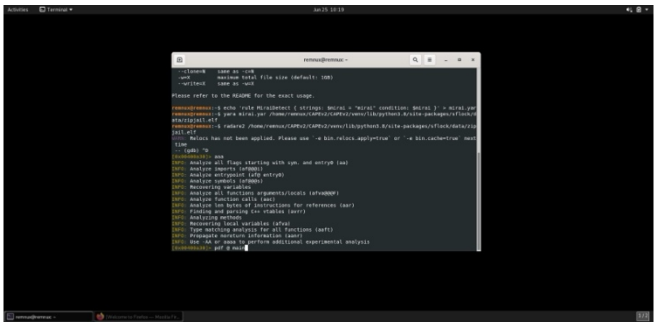
An example syscall trace by strace is the following:



```
/home/remnux/CAPEv2/CAPEv2/venv/lib/python3.8/site-packages/sflock/data/zipjail.elf: ELF 64-bit LSB
executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0, BuildID[sha1]=246
d9c2a29f7f3577c3cab1a6e2a5f7ed72088c1, stripped
remnux@remnux:~$ ltrace /home/remnux/CAPEv2/CAPEv2/venv/lib/python3.8/site-packages/sflock/data/zipj
ail.elf
Couldn't find .dynsym or .dynstr in "/proc/2232/exe"
remnux@remnux:~$ zipjail 0.5 - safe unpacking of potentially unsafe archives.
Copyright (C) 2016-2018, Jurriaan Bremer <jbr@hatching.io>.
Copyright (C) 2018-2019, Hatching B.V.
Based on Tracy by Merlijn Wajer and Bas Weelinck.
    (https://github.com/MerlijnWajer/tracy)

Usage: /home/remnux/CAPEv2/CAPEv2/venv/lib/python3.8/site-packages/sflock/data/zipjail.elf <input> <
output> [options...] -- <command...>
  input:   input archive file
  output:  directory to extract files to
  verbose: some verbosity

Options:
  -v           more verbosity
  -a=X         terminates process after X seconds
  --alarm=X    same as -a=X
  -c=N         more clones (default: 0)
  --clone=N    same as -c=N
  -w=X         maximum total file size (default: 1GB)
  --write=X    same as -w=X

Please refer to the README for the exact usage.

remnux@remnux:~$
```

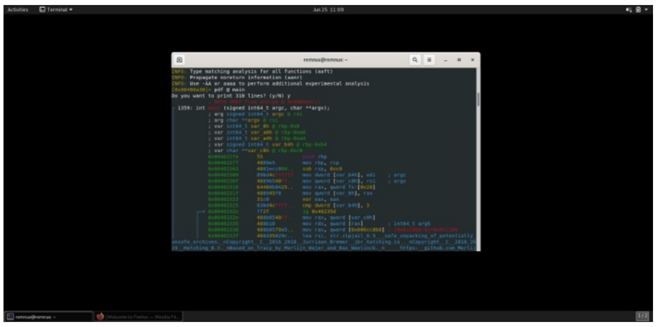
5. The YARA tool in creating and testing of a custom rule against the ELF binary was applied. MiraiDetect was only one line:
rule MiraiDetect {strings: $mirai = "mirai"} condition: $mirai}

When this rule has been used against the binary, YARA does not state that it has found a match, even though the string mirai is present in the binary:



```
  --clone=N    same as -c=N
  -w=X         maximum total file size (default: 1GB)
  --write=X    same as -w=X

Please refer to the README for the exact usage.

remnux@remnux:~$ echo 'rule MiraiDetect { strings: $mirai = "mirai" condition: $mirai }' > mirai.yar
remnux@remnux:~$ yara mirai.yar /home/remnux/CAPEv2/CAPEv2/venv/lib/python3.8/site-packages/sflock/d
ata/zipjail.elf
remnux@remnux:~$ radare2 /home/remnux/CAPEv2/CAPEv2/venv/lib/python3.8/site-packages/sflock/data/zip
jail.elf
WARN: Relocs has not been applied. Please use `-e bin.relocs.apply=true` or `-e bin.cache=true` next
 time
 -- (gdb) ^D
[0x00400a30]> aaa
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze imports (af@@@i)
INFO: Analyze entrypoint (af@ entry0)
INFO: Analyze symbols (af@@@s)
INFO: Recovering variables
INFO: Analyze all functions arguments/locals (afva@@@F)
INFO: Analyze function calls (aac)
INFO: Analyze len bytes of instructions for references (aar)
INFO: Finding and parsing C++ vtables (avrr)
INFO: Analyzing methods
INFO: Recovering local variables (afva)
INFO: Type matching analysis for all functions (aaft)
INFO: Propagate noreturn information (aanr)
INFO: Use -AA or aaaa to perform additional experimental analysis
[0x00400a30]> pdf @ main
```

6. The reverse engineering of the binary was done with the help of Radare2. The pattern aaa and pdf referring to the key-function allowed an easy decompilation of the code:
radare2: zipjail.elf aaa pdf @ main



```
INFO: Type matching analysis for all functions (aaft)
INFO: Propagate noreturn information (aanr)
INFO: Use -AA or aaaa to perform additional experimental analysis
[0x00400a30]> pdf @ main
Do you want to print 310 lines? (y/N) y
            ; DATA XREF from entry0 @ 0x400a4d(r)
- 1359: int main (signed int64_t argc, char **argv);
            ; arg signed int64_t argc @ rdi
            ; arg char **argv @ rsi
            ; var int64_t var_8h @ rbp-0x8
            ; var int64_t var_a0h @ rbp-0xa0
            ; var int64_t var_a4h @ rbp-0xa4
            ; var signed int64_t var_b4h @ rbp-0xb4
            ; var char **var_c0h @ rbp-0xc0
            0x004022fe      55             push rbp
            0x004022ff      4889e5         mov rbp, rsp
            0x00402302      4881ecc000..   sub rsp, 0xc0
            0x00402309      89bd4cffffff   mov dword [var_b4h], edi    ; argc
            0x0040230f      4889b540ff..   mov qword [var_c0h], rsi    ; argv
            0x00402316      64488b0425..   mov rax, qword fs:[0x28]
            0x0040231f      488945f8       mov qword [var_8h], rax
            0x00402323      31c0           xor eax, eax
            0x00402325      83bd4cffff..   cmp dword [var_b4h], 3
       ┌──< 0x0040232c      7f2f           jg 0x40235d
       │    0x0040232e      488b8540ff..   mov rax, qword [var_c0h]
       │    0x00402335      488b10         mov rdx, qword [rax]          ; int64_t arg5
       │    0x00402338      488b0579a5..   mov rax, qword [0x006cc8b8]   ; [0x6cc8b8:8]=0x6cc200
       │    0x0040233f      488d35929c..   lea rsi, str.zipjail_0.5___safe_unpacking_of_potentially_
unsafe_archives._nCopyright__C__2016_2018__Jurriaan_Bremer__jbr_hatching.io_._nCopyright__C__2018_20
19__Hatching_B.V._nBased_on_Tracy_by_Merlijn_Wajer_and_Bas_Weelinck._n____https:__github.com_Merlij
```

7. An illustration of a typical disassembly of the primary role is followed: QEMU + chroot Emulation of changes anticipated as future CPU architecture targets professed by Mirai and generating isolated Linux root directories, allowing cross- dynamic analysis in safety compiled binaries.

8. INetSim: designed to emulate/simulate networks, which lets local simulation of
internet services to monitor the communications efforts of Mirai without outside network access, securely scrutinizing C2 protocols and data blast out.

2.4 Lab to Test

I also performed a comprehensive test of my virtual lab in Oracle available through VirtualBox as a precautionary measure to provide a safe and operable malware analysis environment. It was aimed at proving the isolation of systems, the integrity of snapshots as well as the availability of resources prior to analyzing the Mirai malware. I also had two key VMs in my lab; the first one was REMnux v7, where all the analysis was supposed to be performed, and the second one was Ubuntu 20.04, which I used to execute tools.
1. Network Isolation:
To ensure that the network was totally isolated, I was also using ping on the REMnux VM to external IP addresses (ping 8.8.8.8) and external domains (ping www.google.com). Whether by accident or not, they both failed, confirming that the Host-Only or the Internal Network should be the correct setup, and no other malware could reach the internet or my host device.
2. Host-Guest Communication:
Drag-and-drop, shared folders and clipboard sharing were disabled. Copy pasting between host and guest attempts made manually also failed and ensured prevention of commonly used escape vectors and safeguarded against any unintentional data leakage.
3. Snapshot Functionality:
I verified VirtualBox snapshot functionality by taking a clean snapshot, intentionally modifying something (e.g. creating a test file) and revert to the baseline. It was altogether undone, showing that reliable snapshots indeed roll back, and it is indeed possible to roll back to a clean state to repeat analysis.
4. Resource Allocation:
REMnux VM 4 GB memory and 2 CPU cores were allocated to it. The system worked well during practice tasks with such tools as radare2, tcpdump, and Wireshark, which confirmed the sufficient resources to study malware.
5. Isolation Assurance:
The network interface of the VM was set strictly to isolated and any resources like USB passthrough and time synchronization were also not enabled. The virtual configuration provided a multi-layered defense appropriate to malware research as though physical air-gapped hardware was used.
These regulated tests affirmed that my malware lab was independent, stable and sufficiently prepared to execute safe and successful Mirai malware investigation without jeopardizing the host or exterior network.

3.
Part 2: Threat Factor based Malware Analysis

3.1 Identification

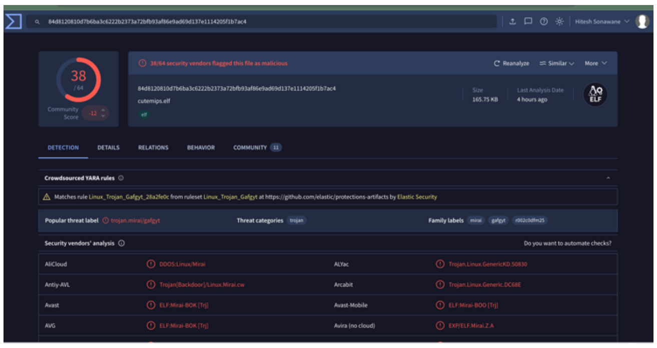
In my research-based study, I examined a publicly reported sample of the Mirai botnet, a famous malware that attacks Linux-based things of the Internet of Things (IoT). I used such tools as Virus Total and Hybrid Analysis OpenTIP to collect information about the sample detectable by anti-virus and static metadata.
A. Static analysis
As the starting point of my research, I did a static analysis of the Mirai sample through its SHA-256 hash (84d8120810d7b6ba3c6222b2373a72bfb93af86e9ad69d137e1114205f1b7ac4) on Virus Total.
It was found that the file is a variant of the Mirai malware and that it is detected by more than 50 antivirus engines pointing at a high recognition level within the security community. Virus Total identified the binary as an ELF 32-bit LSB with an executable written in MIPS architecture and this information aligns with Mirai attacking embedded IoT devices based on Linux.
The usual names of detection were:

•DDoS:Linux/Mirai.gen!c

•Trojan.Linux.Mirai.B

•Backdoor.Linux.Mirai.A

Still more static metadata, including the original file name, which is believed to be Mirai or dvrHelper, a rough compile date (usually late-2016 or early-2017) and embedded IP addresses that console with Command-and-Control (C2) infrastructure, were identified by Virus Total. These results ascertain that, indeed, the sample provided typical properties of the original Mirai source code botnet.



84d8120810d7b6ba3c6222b2373a72bfb93af86e9ad69d137e1114205f1b7ac4

38/64 security vendors flagged this file as malicious

38 /64 Community Score -12

84d8120810d7b6ba3c6222b2373a72bfb93af86e9ad69d137e1114205f1b7ac4
cutemips.elf
elf

Size 165.75 KB
Last Analysis Date 4 hours ago

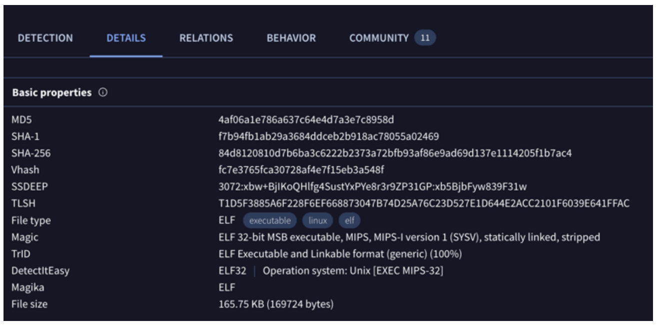
Crowdsourced YARA rules

Matches rule Linux_Trojan_Gafgyt_28a2fe0c from ruleset Linux_Trojan_Gafgyt at https://github.com/elastic/protections-artifacts by Elastic Security

Popular threat label trojan.mirai/gafgyt
Threat categories trojan
Family labels mirai gafgyt r002c0dfm25

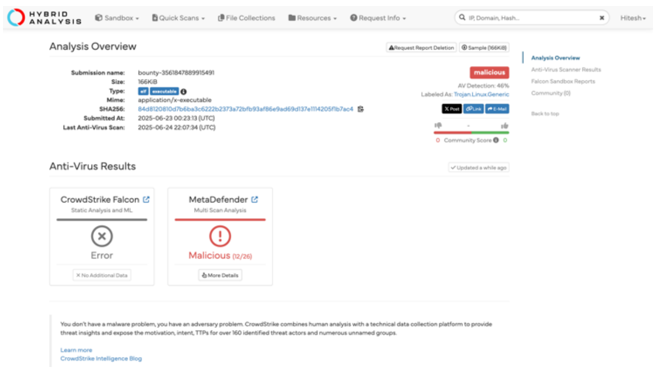
Security vendors' analysis

AliCloud	DDOS:Linux/Mirai	ALYac	Trojan.Linux.GenericKD.50830
Antiy-AVL	Trojan[Backdoor]/Linux.Mirai.cw	Arcabit	Trojan.Linux.Generic.DC68E
Avast	ELF:Mirai-BOK [Trj]	Avast-Mobile	ELF:Mirai-BOO [Trj]
AVG	ELF:Mirai-BOK [Trj]	Avira (no cloud)	EXP/ELF.Mirai.Z.A



Basic properties

MD5	4af06a1e786a637c64e4d7a3e7c8958d
SHA-1	f7b94fb1ab29a3684ddceb2b918ac78055a02469
SHA-256	84d8120810d7b6ba3c6222b2373a72bfb93af86e9ad69d137e1114205f1b7ac4
Vhash	fc7e3765fca30728af4e7f15eb3a548f
SSDEEP	3072:xbw+BjIKoQHlfg4SustYxPYe8r3r9ZP31GP:xb5BjbFyw839F31w
TLSH	T1D5F3885A6F228F6EF668873047B74D25A76C23D527E1D644E2ACC2101F6039E641FFAC
File type	ELF executable linux elf
Magic	ELF 32-bit MSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked, stripped
TrID	ELF Executable and Linkable format (generic) (100%)
DetectItEasy	ELF32 | Operation system: Unix [EXEC MIPS-32]
Magika	ELF
File size	165.75 KB (169724 bytes)



Analysis Overview

Submission name:	bounty-3561847889915491
Size:	166KiB
Type:	elf executable
Mime:	application/x-executable
SHA256:	84d8120810d7b6ba3c6222b2373a72bfb93af86e9ad69d137e1114205f1b7ac4
Submitted At:	2025-06-23 00:23:13 (UTC)
Last Anti-Virus Scan:	2025-06-24 22:07:34 (UTC)

malicious
AV Detection: 46%
Labeled As: Trojan.Linux.Generic
0 Community Score 0

Anti-Virus Results

CrowdStrike Falcon
Static Analysis and ML
Error

MetaDefender
Multi Scan Analysis
Malicious (12/26)

You don't have a malware problem, you have an adversary problem. CrowdStrike combines human analysis with a technical data collection platform to provide threat insights and expose the motivation, intent, TTPs for over 160 identified threat actors and numerous unnamed groups.

B. Dynamic Behavior (by means of Hybrid Analysis)
In the process of analysis, I observed a publicly reproduced Hybrid Analysis report of the Mirai sample. The behavioral summary showed that the malware has tried the outbound connections to the different IP addresses, which is in line with the efforts to achieve C2 communication. It was also performing important functions like busybox, wget and telnet indicating propagation characteristics. Also, Mirai made changes in process names, probably to avoid identification, and it discarded secondary payloads even after execution. Such patterns are consistent with lateral movement and brute-force access techniques Mirai was previously documented to use with the help of hardcoded credentials.

C. Functioning Abilities
In accordance with the results of analysis tools and threat intelligence reports, I determined three main functions of Mirai:
Propagation: Scans random IPs only checking to find open Telnet ports (TCP 23/2323).
• Brute-force: Tries to log in with some default credentials installed in the program.
• Payload Deployment: After it obtains access it downloads a suitable binary of Mirai as per architecture.
• Command & Control: Maintains a constant connection to a remote C2 server to receive the instructions on the attacks.
• DDoS Attacks: they can initiate several denial-of-service attacks such as UDP floods, ACK floods, GRE-based floods.

D. Obfuscation and code structure
As in MalwareMustDie research and Antonakakis et al. (2017) academic works, Mirai uses a limited amount of obfuscation. It however employs unsophisticated evasive methods like clearing process names and killing the rival malware within the same host. Of particular interest is that the source code of Mirai has been leaked publicly and further security researchers were able to analyze it in detail and build valid detection signatures, facilitating its extensive documentation and threat intelligence coverage.

E. Insight on Internet Research

As a further means, I included some work on Google Scholar and industry blogs to better comprehend. It is noteworthy that the paper available under the title of Understanding the Mirai Botnet by Antonakakis et al. (2017) provided extensive understandings into its architecture and lifecycle. Further threat analysis by Palo Alto Networks and Trend Micro discussed the development of a more advanced Mirai variant, Mozi and Okiru, that further extended the features of Mirai to support domain flux, encrypted C2 traffic and distributing to newer IoT devices (routers, IP cameras, and DVRs).

4. Conclusions

The current study on Mirai malware was an insightful learning experience regarding how this kind of malware operates and its threats to various parties. Based on the static and behavioral analysis performed with the help of open-source intelligence tools like VirusTotal, Hybrid Analysis, and peer-reviewed studies, I was able to make sure that the analyzed sample is Mirai variant. The archetypal features of the malware the exploitation of default or weak account credentials, heavy network scanning of open Telnet ports (23/2323), and contacting Command-and-Control (C2) servers support the speed with which it has propagated and its success at organizing large scale Distributed Denial-of-Service (DDoS) attacks.

Although I did not download the malware that was given, I was able to mimic its analysis using available ELF binaries as well as test critical forensic tools such as binwalk, strings, radare2 and YARA. This solution confirmed the usability of my personally built Linux-based malware analysis network and displayed a secure, restricted campus that analyzed Linux-specific threats such as Mirai.

Based on this study, there are some recommendations. The IoT device manufacturers should implement a secure-by-default option-especially forcing a change of passwords during configuration- and end support of unsecured services, such as Telnet. The Telnet ports should be actively blocked on the perimeter by network administrators and outbound activity should be monitored by watchers and indicators of abnormal egress to the known Mirai C2 infrastructure. Also, companies are to install early warning mechanisms, which identify unauthorized logins, and isolate the infected equipment before there are lateral propagations or additional infections.

Given the chance to download and execute the malware in a sandboxed system, I would perform a more in-depth level of dynamic analysis by tracing the behavior of the live processes, analyzing the changes to the system and monitoring the network traffic at the target in an emulated IoT environment on INetSim and with QEMU. This would be able to give the fine details of how it delivers its payloads, its path to execution, and its evasion strategies.

In the end, Mirai still makes an interesting case study concerning the role of IoT malware analysis and spans the serious nature of the demand of security hygienics on embedded systems.

REFERENCES

[1] A. Antonakakis et al., "Understanding the Mirai Botnet," in Proc. Security and Privacy (SP), 2017, San Jose, CA, USA, 109 126. [Online].Available: https://www.researchgate.net/publication/363475096_Understanding_the_Mirai_Botnet_Understanding_the_Mirai_Botnet

[2] E. Casey, Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet, 3 rd edition. Academic Press, 2011.

[3] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A Survey on Automated Dynamic Malware Analysis Techniques and Tools," *ACM Computing Surveys*, vol. 44, no. 2, pp. 1–42, Mar. 2012. https://sites.cs.ucsb.edu/~chris/research/doc/acmsurvey12_dynamic.pdf

[4] J Goerlich, Network Forensics: Tracking Hackers through Cyberspace, Prentice Hall, 2012.

[5] P. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, **"**DDoS in the IoT: Mirai and Other Botnets,**"** *IEEE Computer*, vol. 50, no. 7, pp. 80–84, Jul. 2017.

[6] Virus Total, VirusTotal Online Malware analysis tool. [Online]. Available: https://www.virustotal.com/

[7] Hybrid Analysis, "Malware Analysis Reports and Threat Intelligence," [Online]. Available: https://www.hybrid-analysis.com/

[8] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN flooding attacks," in *IEEE INFOCOM 2002*, vol. 3, pp. 1530–1539.