Tamer: A Sandbox for Facilitating and Automating IoT Malware Analysis with Techniques to Elicit Malicious Behavior

Shun Yonamine, Yuzo Taenaka and Youki Kadobayashi
Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Nara 630-0192, Japan

Keywords: Sandbox, IoT, Malware, Dynamic Analysis, Automated Analysis, ARM.

Abstract: As malware poses a significant threat to IoT devices, the technology to combat IoT malware, like sandbox, has not received enough attention. The majority of efforts in existing researches have focused on x86-flavored binaries that are not used for IoT devices. In fact, we have witnessed that many samples of IoT malware that can be observed in the wild are ARM binaries. In this paper, we propose a novel sandbox for analyzing Linux malware including IoT malware. Our sandbox system, called Tamer, supports dynamic analysis for ARM binaries and has some features to automate and facilitate IoT malware analysis, like the automated interaction mechanism and the fake network environment for dynamic analysis. In addition, our system adopts features, like dynamic binary instrumentation and virtual machine introspection, which may allow retrieving further insights from malware. With the dataset of real-world malware, we demonstrated that our sandbox system can analyze IoT malware which is specifically designed for infecting IoT devices. Through an analysis experiment on a large number of IoT malware samples, we demonstrate a possibility that our system could facilitate a large scale analysis in an automated manner and retrieve further insights from IoT malware.

1 INTRODUCTION

With the spread of IoT devices, IoT malware has been increasingly sophisticated time by time. Some recent studies have promoted awareness of the threats from IoT malware (Cozzi et al., 2018; Cozzi et al., 2020; Carrillo-Mondéjar et al., 2020). Sandbox is one of the malware analysis methods that enable extracting the details of malware behavior. As sandbox lets malware run on a real system, it is useful to investigate various malware behavior on certain conditions of the system. In addition to this feature, sandbox attracts attention as the solution of analyzing a large number of malware samples systematically and automatically. These features are expected to contribute to following the trend of IoT malware growing and changing quickly.

In past, researchers have proposed novel sandbox systems that are capable of automated dynamic malware analysis, and each of them handles specific use-cases (Cuckoo, 2013; Willems et al., 2007; Bayer et al., 2006; Monnappa, 2015). However, the majority of previous works have focused on analyzing conventional malware, and the sandbox for IoT malware has not received enough attention and not been explored in detail. In particular, existing sandbox systems (Cuckoo, 2013; Monnappa, 2015) mainly target on x86 architectures but do not support architectures like ARM that are majorly used by IoT. While methods for facilitating dynamic analysis have matured through previous studies, these are mainly adjusted for x86 architectures. Since the execution environment of ARM is different from x86 architectures, it requires additional features to facilitate IoT malware analysis.

In this paper, we propose a novel sandbox, called Tamer, for analyzing IoT malware. Different from existing sandboxes which mainly target on x86 architectures, our sandbox system supports architectures that are used by IoT devices, such as ARM. As a feature to elicit network behavior from the target malware sample, we construct a fake network environment where a sandbox machine and pseudo-C&C server are connected. The fake C&C server allows observing network data that could be exchanged between malware and the C&C server. Further, by leveraging `expect`[1] utility, Tamer automatically operates the Linux machine where is the execution environment of IoT malware. In addition, Tamer supports advanced binary techniques like dynamic binary in-

[1]https://linux.die.net/man/1/expect

Yonamine, S., Taenaka, Y. and Kadobayashi, Y.
Tamer: A Sandbox for Facilitating and Automating IoT Malware Analysis with Techniques to Elicit Malicious Behavior.
DOI: 10.5220/0010968300003120
In *Proceedings of the 8th International Conference on Information Systems Security and Privacy (ICISSP 2022)*, pages 677-687
ISBN: 978-989-758-553-1; ISSN: 2184-4356

strumentation (DBI) and virtual machine introspection (VMI).

For evaluation, we conduct several malware analysis experiments on the dataset of real-world malware samples. We show that our sandbox system promotes deeper details about the malware behavior. The evaluation supports evidence that how Tamer sandbox can serve as a choice for IoT malware analysis.

This paper makes the following contributions:

- We propose a novel sandbox, called Tamer, for analyzing IoT malware. Tamer supports analyzing binaries for ARM, an architecture mostly used in IoT devices. In addition, Tamer has key features to facilitate IoT malware analysis: The fake network is a dedicated environment to facilitate analyzing the network behavior of malware using the fake C&C server. The auto-manipulation mechanism using the `expect` allows to performing behavior analysis in an automated manner. Moreover, to the best of our knowledge, Tamer is the first sandbox that attempts to combine advanced binary analysis techniques such as DBI and VMI.
- Through the experimental evaluation on the dataset of real-world malware, we demonstrated that our system can analyze IoT malware which is dedicated to infecting IoT devices. Moreover, the result of an analysis experiment on a large volume of samples suggests that our system can analyze a huge amount of IoT malware samples in an automated manner, and may highlight recent trends in IoT malware families.
- We will release the details of the implementation of Tamer as open-source, and the list of md5 hashes of malware samples in the dataset, at the following link[2]. We expect that some outcomes we observed through analyzing a large number of samples could be interesting to security researchers. This is for promoting to replicate our experiments and obtain the same observations.

2 RELATED WORKS

2.1 Observing Landscape of IoT Malware

Why the ARM based Linux Malware Matters: In this study, we focus on the Linux malware for the ARM architecture since this architecture is popular for consumer IoT devices and commonly targeted by IoT malware. In fact, our position can be supported by some studies. In the recent study (Cozzi et al., 2020), they explored a large number of malware samples that have been submitted to VirusTotal over a period of almost four years (from January 2015 to August 2018). In their dataset, the ARM 32-bit malware accounted for the largest number of samples, 39.05% of the total 93,652 samples.

Furthermore, according to the VirusShare dataset (VirusShare, 2020), a repository of malware samples, the ARM architecture accounted for the majority of Linux malware samples collected in the recent period (from Feburary 2019 to April 2020). In detail, from our survey on the dataset referred to `VirusShare_ELF_20200405`, the ARM accounted for one-third or 13,963 out of a total of 43,553 samples. In short, based on these observations, we claim that the anti-malware methodology that focuses on ARM malware is worth considering.

2.2 Existing Sandbox Systems

To date, various sandbox systems have been proposed and each of these focuses on various use-cases to combat the malware. In general, the purpose of sandboxes has focused on behavioral analysis and the information obtained from the analysis is used for updating intrusion detection system's signatures or removing malware from an infected machine, and so on (Willems et al., 2007). Although various methods have been explored as sandbox systems have evolved, we have witnessed that sandbox systems for analyzing IoT malware have not received enough attention.

As an example, Willems et al. designed CWSandbox (Willems et al., 2007), a sandbox that aims to generate the machine-readable report to initiate automated responses. However, their sandbox is for analyzing malware targeting Windows systems, not IoT malware. In addition, as the authors admitted that CWSandbox might cause some harm to other machines connected to the network, their sandbox does not have some tricks to make the sandbox separated from the Internet.

Regarding a limitation of CWSandbox, here we could suggest that the network setting should be paid more cautious in IoT malware analysis. The reason is, in many cases, network activity by IoT malware involves destructive functions (e.g. DDoS attack, Brute-force attack to propagate through SSH/Telnet). Therefore, to construct a dedicated network setting for malware analysis, it is also necessary to set up a server that serves as a listener for the C&C connection.

Bayer et al. proposed Anubis (Bayer et al., 2006), a novel sandbox for automated malware analysis. Un-

[2]https://github.com/shun-yo/Tamer

Table 1: Setting for 32-bit ARM Linux.

Hardware	ARM Versatile/PB(ARM926EJ-S)
Architecture	ARM v5
Kernel	Linux 3.2.1
Memory	256MB
Virtual Disk	debian_squeeze_armel_standard.qcow2 (Debian.org, 2014)

Table 2: System specification inside Tamer.

	OS	CPU	Memory
Host machine	Ubuntu14.04(x86_64)	Intel Xeon Silver @1.80GHz*4	16GB
Sandbox machine	32-bit ARM Linux (kernel ver: 3.2.1)	QEMU	256MB
Fake C&C server	Linux (Debian 9.12 i386)	QEMU	1 GB

fortunately, their sandbox is for analyzing Windows malware. Besides, their work is not motivated to monitor network behavior. As with Willems et al., their study does not care about the network environment for monitoring network behavior by malware. In detail, while the purpose of their sandbox is to understand functionalities of a given malware sample, their aim is to get the knowledge about the functionality of malware for removal.

While CWSandbox (Willems et al., 2007) and Anubis (Bayer et al., 2006) focus their scope on Windows malware, Limon (Monnappa, 2015) is a sandbox for analyzing Linux malware. Limon has a minimal mechanism for observing network behavior. It uses simulated network services with the `inetsim` (inetsim, 2020) on the analysis machine and allows monitoring network behavior by malware (like they analyzed Tsunami (Monnappa, 2015)). However, Limon does not support analyzing ARM binaries. In detail, analysis methods used by Limon leverage features of VMware that are for x86 architectures, and are not applicable to ARM. Similarly, Cuckoo (Cuckoo, 2013) is an open-source sandbox, but it does not support analyzing ARM binaries.

Overall, to the best of our knowledge, at the time of writing, sandbox systems for IoT malware have not received enough attention. Even though several works aim to handle Linux malware that could relate to IoT, they do not take into account some features that are necessary to facilitate IoT malware analysis (e.g. ARM support, dedicated network settings).

3 SYSTEM OVERVIEW OF TAMER SANDBOX

First of all, to be able to run and analyze ARM binaries, we need dedicated platforms for ARM, including CPU and OS. Thus, we choose to use QEMU emulator to run a dedicated ARM based Linux on an emulated CPU. For 32-bit ARM Linux machine, we use Debian Linux where the kernel was cross-compiled for ARM architecture, as we describe in Table 1. In our setting, we use `qemu-system-arm` in accordance with options "`-M versatilepb`" to emulate ARM board for general-purpose Linux target. To analyze the dynamic behavior of IoT malware, our proposed system, Tamer, has three components. There are, 32-bit ARM Linux machine for a sandbox where a target sample is executed, the fake network where a sandbox machine locates, an analysis tool to analyze the execution trace log which can be obtained after executing malware is finished. The execution trace log records the full system activities and allows replaying it for analysis. To support analysis using execution trace log, our system leverages PANDA (Dolan-Gavitt et al., 2015), a framework that is built upon the QEMU emulator, while we slightly modified it for our purpose. Note that PANDA supports dynamic binary instrumentation (DBI) and virtual machine introspection (VMI). Thus, the current design of Tamer leaves analysts an option to use advanced binary analysis techniques like taint analysis.

The fake network is a network whose setting is controlled for malware analysis. This is a host-only network and mainly serves two purposes. First, by setting it as a closed network, it promises to prevent attacking packets from going outside the network. This feature is preferable to maintain safety for malware analysis. Second, since network routing in the fake network is controllable by `iptables`, it allows redirecting network connections from the sandbox to the fake C&C server we set up. In short, these features of the fake network allow observing network behavior of malware in a safer manner, besides the fake C&C server serves functionality to elicit network behavior of malware. Finally, we show the specifications of the machines inside our system in Table 2.

Figure 1 shows that how our system works by combining those components described above. In our system, first, a target malware sample is put into the sandbox machine or Linux. Then, the malware is launched during a fixed timeout (currently set 3 minutes) and the execution trace log is retrieved. To be more precise, our proposed system allows observing network behavior by using the fake C&C server even if a C&C server where the target is supposed to connect has already been closed. Finally, our analysis tool analyzes execution trace and obtains dynamic information of malware.

In the rest of this section, we describe details to make our system follow our concepts.

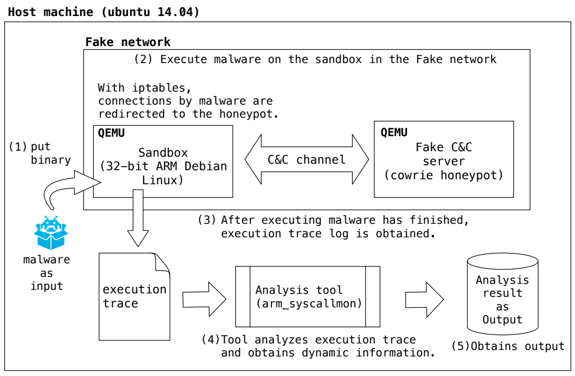

Host machine (ubuntu 14.04)
Fake network
(2) Execute malware on the sandbox in the Fake network
With iptables, connections by malware are redirected to the honeypot.
QEMU
Sandbox (32-bit ARM Debian Linux)
C&C channel
QEMU
Fake C&C server (cowrie honeypot)
(1) put binary
malware as input
(3) After executing malware has finished, execution trace log is obtained.
execution trace
Analysis tool (arm_syscallmon)
Analysis result as Output
(4)Tool analyzes execution trace and obtains dynamic information.
(5)Obtains output

Figure 1: Inside Tamer and analysis flow.

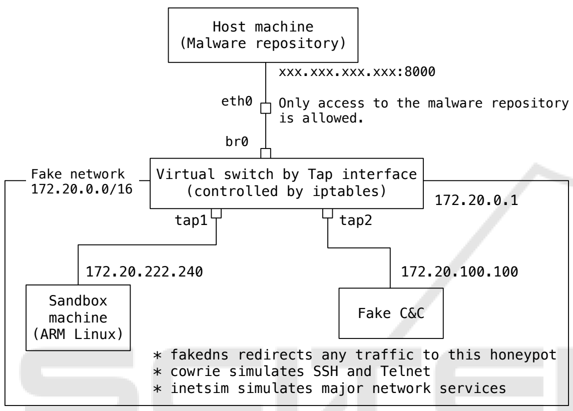

Host machine (Malware repository)
xxx.xxx.xxx.xxx:8000
eth0
Only access to the malware repository is allowed.
br0
Fake network 172.20.0.0/16
Virtual switch by Tap interface (controlled by iptables)
172.20.0.1
tap1
tap2
172.20.222.240
172.20.100.100
Sandbox machine (ARM Linux)
Fake C&C
* fakedns redirects any traffic to this honeypot
* cowrie simulates SSH and Telnet
* inetsim simulates major network services

Figure 2: Fake network to emulate C&C server.

3.1 Fake Network Environment

The fake network is a controlled network environment for malware analysis and isolated from the Internet. There are two advantages to use the fake network. First, it prevents connections by malware to propagate infection from going outside the network. This is for conducting IoT malware analysis in a safer manner. Second, it redirects connections from the sandbox to the fake C&C server. This is for inspecting the malware's network behavior. Commonly, malicious behavior of malware relies on C&C communication therefore analyzing the network behavior is important to understand malware. To make the fake network isolated and accessible from a sandbox machine, we carefully designed the routing setting with `iptables`. The network in which the target malware is executed is illustrated in Figure 2.

In order to set up the fake network, we first create a virtual bridge using `brctl` (LinuxFoundation, 2020) utility as shown in Figure 2. Then we create two TUN/TAP interfaces, namely `tap1` and `tap2`, and bind both of them to the `br0`. This setting allows the sandbox machine to access the malware repository which is located outside the Fake network. We note that this setting is for putting a target sample into the sandbox machine. Of course, network connections except for access to the malware repository are not permitted.

Table 3: Tools installed on Fake C&C.

Software	Purpose
Cowrie honeypot	SSH/Telnet listener
fakedns	Respond to DNS query with fake IP addr
inetsim	Emulate common network services

On the other hand, the fake C&C server emulates network services to handle requests from the sandbox where a malware sample has launched. As described above, network connections from the sandbox are redirected to the fake C&C server. Table 3 shows software or tools that are installed on the Fake C&C server. To handle SSH/Telnet connections that are commonly requested by IoT malware, we use Cowrie (Michel Oosterhof, 2014) honeypot which serves as SSH/Telnet server. In particular, the majority of IoT malware attempts to propagate through SSH or Telnet. Moreover, the proposed system leverages `inetsim` to emulate common network services. This aims to handle requests that may be sent by malware. Moreover, `fakedns` respond to DNS queries with the IP address of the fake C&C server. This allows redirecting network connections using domain names.

3.2 Automated Interaction Mechanism

The proposed system has a mechanism to automatically manipulate the sandbox machine, in this case ARM Linux. The manipulation can be programmed by a user. We developed a script to achieve this mechanism. Specifically, the script we developed leverages the `expect` to automate interaction with the sandbox. With this script, we manipulate the sandbox in an automated manner.

The aim of this feature is, for malware analysis, it is necessary to operate the sandbox where a target malware is to be launched. To be more precise, it needs at least 2 steps in the sandbox. Those steps are namely, (1)putting a target sample in the sandbox at first, and then (2)executing it, as shown in Figure 1. In order to manipulate the sandbox, we leverage `sendkey` which is, a command which can be utilized through the QEMU console[3]. This command invokes keyboard events in the guest machine. With this command, we can manipulate the sandbox or Linux through keyboard operation.

[3]https://en.wikibooks.org/wiki/QEMU/Monitor

```
wget http://<repository's ipaddr>:8000/<FILENAME>
     -O ./iotmal
chmod +x iotmal
./iotmal
```

List 1: Example of steps to launch malware for dynamic analysis.

```
(qemu) begin_record iotmal_<FILENAME>_<DATETIME>
...wait for 3 minutes...
(qemu) end_record
```

List 2: Operation through QEMU console to save the execution trace log for PANDA.

3.3 Automating Steps to Launch Malware in the Sandbox

As described before, the automated interaction mechanism is one of the key features of our system. This is actually an automatic interaction mechanism using the expect, which can be attached from external the QEMU. In other words, this is an ad-hoc solution and easy to attach. From that point, this feature is different from existing approaches that rely on the control by an agent installed in the guest machine. In the following, we describe the advantage of our system. The advantage is that our system allows automating several steps that are needed to be performed on starting malware analysis. These steps are, for example, putting a target binary into the sandbox machine and then executing it. We describe these steps in the following.

In the Linux machine, a malware sample is downloaded before being executed. To do this, using the `expect` module, commands (as shown in List 1) are inputted in the Linux machine and executed. In the List, once the sandbox machine downloads a file, it gives the exec permission to the file and executes it.

Further, before the sample is executed, the procedure for logging the execution trace is invoked. After a fixed timeout, currently set to 3 minutes, it stops the logging, as shown in List 2.

3.4 External Tool for Analyzing Dynamic Behavior of Malware

After executing malware in the sandbox is finished, we can obtain the execution trace log. The execution trace log is a file that records the entire OS events that have been invoked in the guest machine. The system events recorded in the execution trace can be replayed on PANDA (Dolan-Gavitt et al., 2015) built upon the QEMU. In short, here we analyze the replayed system events. In order to analyze malware's behavior, we developed a tool, named `arm_syscallmon`, to analyze the execution trace log. We developed this analysis tool as a PANDA plugin. Currently, this prototype focuses the main functionality on examing syscalls that a target malware sample invoked. In detail, `arm_syscallmon` leverages dynamic binary instrumentation (DBI) to monitor instructions that invoke syscall in the guest machine upon the QEMU. On the other hand, to identify the process of target malware, it leverages virtual machine introspection (VMI) to reference `task_struct` data in the kernel.

Table 4: Files accessed by IoTReaper with `sys_open`.

Path
/tmp/client.run
/dev/watchdog
/dev/misc/watchdog
/etc/hosts
/etc/resolve.conf

4 EXPERIMENTAL EVALUATION

In this section, through several experiments to analyze the behavior of IoT malware, we demonstrate the capability of Tamer from multiple perspectives. In short, we show that Tamer performs features as follows:

- It allows retrieving information for understanding dynamic characteristics of malware.
- It allows analyzing malware which is designed for IoT devices.

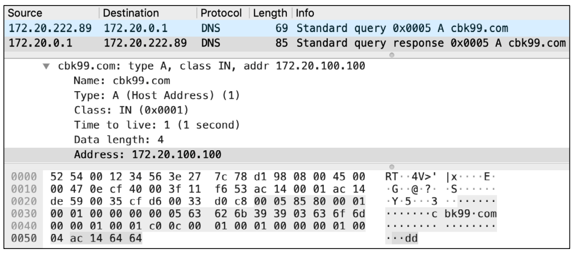
4.1 Capability for IoT Malware Analysis

In order to show the capability of our sandbox, this experiment analyzes a malware sample and retrieves dynamic characteristics or runtime information, such as *syscalls, files, unique strings*, etc. As a sample of IoT malware, we analyzed a real-world malware, IoTReaper[4] (FortiGuard, 2017). This malware is notorious as a variant that targets IoT devices. Besides, the only binaries found were ARM executables.

4.1.1 Use Case: System Call Monitoring

Through system call monitoring, the proposed system obtains the information that relates to the dynamic behavior of the target malware. This use-case aims to understand about dynamic characteristics that the malware will pose once it has launched, e.g., kind of

[4]MD5: ca92a3b74a65ce06035fcc280740daf6



```
Source          Destination     Protocol  Length  Info
172.20.222.89   172.20.0.1      DNS           69  Standard query 0x0005 A cbk99.com
172.20.0.1      172.20.222.89   DNS           85  Standard query response 0x0005 A cbk99.com

    ▼ cbk99.com: type A, class IN, addr 172.20.100.100
        Name: cbk99.com
        Type: A (Host Address) (1)
        Class: IN (0x0001)
        Time to live: 1 (1 second)
        Data length: 4
        Address: 172.20.100.100

0000  52 54 00 12 34 56 3e 27  7c 78 d1 98 08 00 45 00   RT··4V>' |x····E·
0010  00 47 0e cf 40 00 3f 11  f6 53 ac 14 00 01 ac 14   ·G··@·?· ·S······
0020  de 59 00 35 cf d6 00 33  d0 c8 00 05 85 80 00 01   ·Y·5···3 ········
0030  00 01 00 00 00 00 05 63  62 6b 39 39 03 63 6f 6d   ·······c bk99·com
0040  00 00 01 00 01 c0 0c 00  01 00 01 00 00 00 01 00   ········ ········
0050  04 ac 14 64 64                                     ···dd
```

Figure 3: Fakedns returns IP address of Fake C&C.

syscalls, files that the malware will access, and data that will be written into the file or left on the victim's machine. In particular, the data that malware has left on the system can be leveraged in order to detect infection and remove malware from the system (Bayer et al., 2006). Note that behavior analysis is performed on the execution trace log which records the entire system activity. The analysis tool we developed will examine the execution trace log and retrieve information on the syscalls that were invoked during the malware was running.

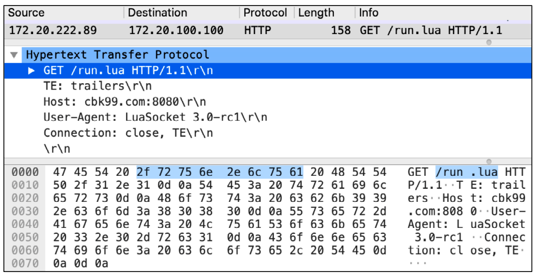
Next, we present some examples of the analysis results obtained from analysis using Tamer. Here, we focus on the arguments of the system calls. For this, Tamer allows retrieving information that indicates characteristics of malware's behavior. As an example, Table 4 shows details of files that a target malware tried to access. It is likely that information on files accessed by malware is beneficial to understand the severity of the threat that the malware poses.

From the output we obtained from the proposed system, we expect that these information can be useful to understand the dynamic characteristic of the target malware. For instance, details of the files accessed by malware brings insights to consider countermeasures against malware (as Cozzi et al. have conducted an investigation to spot the light on the recent trend of Linux malware (Cozzi et al., 2018)). This result shows that our system reveals the runtime behavior invoked IoT malware.

4.1.2 Use Case: Eliciting the Network Fingerprint

This use-case aims to observe the malware's dynamic behavior from the network perspective. The benefit of network behavior analysis is that it allows obtaining network data that are exchanged between malware and the C&C server. In addition, this experiment also supports evidence that the fake C&C elicits the network behavior of malware by acting as a server that receives data sent from malware.

As shown in Figure 4, our sandbox has retrieved network data that the target malware was supposed to

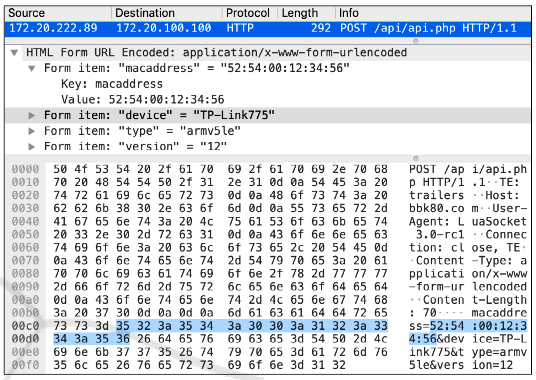


```
Source          Destination      Protocol  Length  Info
172.20.222.89   172.20.100.100   HTTP         158  GET /run.lua HTTP/1.1

▼ Hypertext Transfer Protocol
  ▶ GET /run.lua HTTP/1.1\r\n
    TE: trailers\r\n
    Host: cbk99.com:8080\r\n
    User-Agent: LuaSocket 3.0-rc1\r\n
    Connection: close, TE\r\n
    \r\n

0000  47 45 54 20 2f 72 75 6e  2e 6c 75 61 20 48 54 54   GET /run .lua HTT
0010  50 2f 31 2e 31 0d 0a 54  45 3a 20 74 72 61 69 6c   P/1.1··T E: trail
0020  65 72 73 0d 0a 48 6f 73  74 3a 20 63 62 6b 39 39   ers··Hos t: cbk99
0030  2e 63 6f 6d 3a 38 30 38  30 0d 0a 55 73 65 72 2d   .com:808 0··User-
0040  41 67 65 6e 74 3a 20 4c  75 61 53 6f 63 6b 65 74   Agent: L uaSocket
0050  20 33 2e 30 2d 72 63 31  0d 0a 43 6f 6e 6e 65 63    3.0-rc1 ··Connec
0060  74 69 6f 6e 3a 20 63 6c  6f 73 65 2c 20 54 45 0d   tion: cl ose, TE·
0070  0a 0d 0a                                           ···
```

Figure 4: Network behavior of IoTReaper requests commands from adversary.



```
Source          Destination      Protocol  Length  Info
172.20.222.89   172.20.100.100   HTTP         292  POST /api/api.php HTTP/1.1

▼ HTML Form URL Encoded: application/x-www-form-urlencoded
  ▼ Form item: "macaddress" = "52:54:00:12:34:56"
      Key: macaddress
      Value: 52:54:00:12:34:56
  ▶ Form item: "device" = "TP-Link775"
  ▶ Form item: "type" = "armv5le"
  ▶ Form item: "version" = "12"

0000  50 4f 53 54 20 2f 61 70  69 2f 61 70 69 2e 70 68   POST /ap i/api.ph
0010  70 20 48 54 54 50 2f 31  2e 31 0d 0a 54 45 3a 20   p HTTP/1 .1··TE:
0020  74 72 61 69 6c 65 72 73  0d 0a 48 6f 73 74 3a 20   trailers ··Host:
0030  62 62 6b 38 30 2e 63 6f  6d 0d 0a 55 73 65 72 2d   bbk80.co m··User-
0040  41 67 65 6e 74 3a 20 4c  75 61 53 6f 63 6b 65 74   Agent: L uaSocket
0050  20 33 2e 30 2d 72 63 31  0d 0a 43 6f 6e 6e 65 63    3.0-rc1 ··Connec
0060  74 69 6f 6e 3a 20 63 6c  6f 73 65 2c 20 54 45 0d   tion: cl ose, TE·
0070  0a 43 6f 6e 74 65 6e 74  2d 54 79 70 65 3a 20 61   ·Content -Type: a
0080  70 70 6c 69 63 61 74 69  6f 6e 2f 78 2d 77 77 77   pplicati on/x-www
0090  2d 66 6f 72 6d 2d 75 72  6c 65 6e 63 6f 64 65 64   -form-ur lencoded
00a0  0d 0a 43 6f 6e 74 65 6e  74 2d 4c 65 6e 67 74 68   ··Conten t-Length
00b0  3a 20 37 30 0d 0a 0d 0a  6d 61 63 61 64 64 72 65   : 70···· macaddre
00c0  73 73 3d 35 32 3a 35 34  3a 30 30 3a 31 32 3a 33   ss=52:54 :00:12:3
00d0  34 3a 35 36 26 64 65 76  69 63 65 3d 54 50 2d 4c   4:56&dev ice=TP-L
00e0  69 6e 6b 37 37 35 26 74  79 70 65 3d 61 72 6d 76   ink775&t ype=armv
00f0  35 6c 65 26 76 65 72 73  69 6f 6e 3d 31 32         5le&vers ion=12
```

Figure 5: IoTReaper notifies the adversary once it has been launched in the victim machine.

send to a C&C server. At first glance, these data are intending requesting commands from a C&C server controlled by an attacker (a target sample namely IoTReaper is requesting a Lua script). These data are seemingly characteristic and are expected to be used as network signatures for detection. Similarly, these characteristics can also be seen in the packets used to notify the attacker that the IoTReaper has been launched on the victim machine, as shown in Figure 5. This result shows that the fake C&C server enables handling the connection from malware to retrieve the network data sent from malware. To be more specific, as described before, by crafting network routing with `iptables`, the network connection that a malware initiates is redirected to a fake C&C server. For this, it allows obtaining data sent by malware. In addition, as shown in Figure 3, `fakedns` commits to redirect malware's connections by responding to DNS queries with an IP address of the fake C&C server.

From these results, we have confirmed that our proposed sandbox system has successfully observed network data that the malware generates. Actually, we have obtained data that are characteristic. Besides, the result shows that the honeypot we set up has served as a fake C&C server. Those malicious network behavior are not observable without the effort of the fake C&C server, since in many cases, real C&C servers

Table 5: Dataset (family names are referenced from labels by TrendMicro).

Malware family	Number of samples
Trojan.Linux.MIRAI.SMMR1	100
Backdoor.Linux.GAFGYT.SMMR3	100
Backdoor.Linux.BASHLITE.SMJC	100
Total	300

```
wget http://<repository's ipaddr>:8000/<FILENAME>
     -O ./iotmal
chmod +x iotmal
strace -tt -f -s 2048 -y -o log.txt ./iotmal
curl -T log.txt http://<repository's ipaddr>:8000/
     log_iotmal_<FILENAME>_<DATETIME>.txt
```

List 3: Logging syscall with strace command for large scale analysis.

where the malware connects to are already closed at the time of analysis. In short, the fake C&C server successfully served as a C&C server where the malware was supposed to connect. This result shows that Tamer can retrieve network data generated by malware without the presence of an original C&C server.

4.2 Capability for Automation: Example of Large Scale Analysis

One of the use cases of automated approach in malware analysis is to analyze a large number of samples and understand their population characteristics. In this section, we show that the automation mechanism of our system can analyze trends in IoT malware characteristics.

On the dataset available from VirusShare, which is namely referred to as `VirusShare_ELF_20200405`, we analyzed 100 samples for each family consisting of Mirai, Gafgy, and Bashlite. Table 5 shows the details of our dataset consisting of Mirai, Gafgyt, and Bashlite. According to searching VirusTotal with the MD5 hash of ELF file, TrendMicro labeled each category of samples as shown in the table. In total, we have analyzed 300 samples.

Our analysis experiment is based on analyzing system calls invoked by malware. First, once a target malware sample fetched from our malware repository is executed, syscalls are logged for a fixed timeout (set to 2 minutes). For monitoring system calls, we use `strace`[5], a utility that is appropriate for a large scale analysis from the aspect of time efficiency and the ease of setting. Next, after syscall logging is finished, the output file by `strace` will be sent to the repository to be saved. Then, the sandbox will be restored to the original state using a snapshot. These steps are repeated for all samples. As a result, we can be prepared to analyze system calls for all 300 target samples.

[5]strace: linux syscall tracer. https://strace.io

```
[ [96mBOT JOINED [97m] Arch:  [96mARM4T  [97m||
     Type: LITTLE_ENDIAN]

[*] N0D3 1NF3CT3D | |

\33[0;32mConnected | IP: <IP ADDR> | Type: SERVER
     | Version: Build 2
```

List 4: Example of printable strings in notification banner (Bashlite).

```
POST /cgi-bin/ViewLog.asp HTTP/1.1  Host: <IP ADDR
     >:80  Connection: keep-alive  Accept-Encoding:
      gzip, deflate  Accept: */*  User-Agent:
     python-requests/2.20.0  Content-Length: 227
     Content-Type: application/x-www-form-
     urlencoded     /bin/busybox wget http://<IP
     ADDR>/lockfilebins.sh; chmod +x lockfilebins.
     sh; ./lockfilebins.sh
```

List 5: Example of printable strings in HTTP requests (by Bashlite).

In system call monitoring, we mainly focus on system calls that relate to *file access* and *network activity*. Thus, here we analyze invoked system calls focusing on `open`, `access` and `send`. Our position is that the details of files accessed by the malware are important, since many malicious behaviors involve file activity. Therefore, from the perspective of file access behavior, we examine files opened or used by malware. On the other hand, from the perspective of network activity, we focus on the data that were sent by malware. The reason is that oftenly network data is characteristic and may indicate signs to infer the adversary's intention. In this experiment, the network behaviors we focus on are namely *notifying the successful infection* and *HTTP requests*. This is because these two network behaviors are commonly exhibited by IoT malware and oftenly indicate the printable data that are characteristic. As examples, List 4 and List 5 show the network data that are sent by malware when it has been executed in the victim machine.

To be more specific, our analysis is based on examining arguments of syscalls. To analyze files needed by malware, we examine arguments of `open` and `access`. In particular, malware oftenly invokes `access` to check if a specific file exists in the infected machine. Further, to obtain the network data, we examine `send` to retrieve the data used in an argument.

The results of this experiment highlight what kind of characteristics each family of IoT malware is likely to exhibit. We describe examples as follows: First, Table 6 summarizes files that the samples have ac-

Table 6: Interesting File paths in `open` by Mirai, Gafgyt and Bashlite.

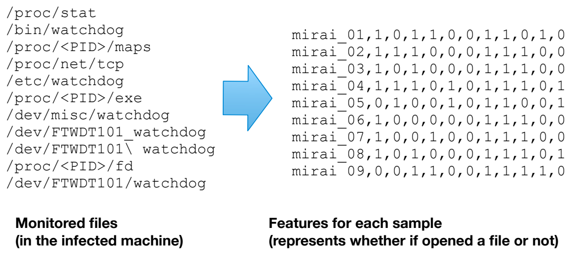
Path	Mirai(%)	Gafgyt(%)	Bashlite(%)
/dev/misc/watchdog	67	80	19
/dev/watchdog	67	80	19
/dev/FTWDT101_watchdog	49	0	0
/proc/	48	4	3
/proc/net/tcp	47	4	3
/dev/FTWDT101 watchdog	43	0	0
/proc/<PID>/fd	30	1	0
/proc/<PID>/exe	25	0	0
/proc/stat	23	0	11
/sbin/watchdog	20	0	0
/proc/<PID>/maps	19	3	3
/dev/watchdog0	15	0	0
/etc/default/watchdog	15	0	0
/dev/FTWDT101/watchdog	12	0	0
/bin/watchdog	11	0	0
/etc/watchdog	9	0	0
/proc/net/route	3	100	83
/etc/ld.so.cache	3	0	12
/etc/rc.d/rc.local	1	6	17
/etc/resolv.conf	0	0	16

Table 7: Files used in `access`.

Path	Mirai(%)	Gafgyt(%)	Bashlite(%)
/usr/bin/python	2	80	43
/etc/ld.so.nohwcap	3	0	12
/etc/resolv.conf	0	0	16
/usr/sbin/telnetd	0	0	1
/usr/bin/apt-get	0	0	2

Table 8: Printable strings found in `send` for network activity.

	Mirai(%)	Gafgyt(%)	Bashlite(%)
Notifies infection	0	12	64
HTTP requests	0	8	21



```
/proc/stat
/bin/watchdog
/proc/<PID>/maps
/proc/net/tcp
/etc/watchdog
/proc/<PID>/exe
/dev/misc/watchdog
/dev/FTWDT101_watchdog
/dev/FTWDT101\ watchdog
/proc/<PID>/fd
/dev/FTWDT101/watchdog
```

```
mirai_01,1,0,1,1,0,0,1,1,0,1,0
mirai_02,1,1,1,0,0,0,1,1,1,0,0
mirai_03,1,0,1,0,0,0,1,1,1,0,0
mirai_04,1,1,1,0,1,0,1,1,1,0,1
mirai_05,0,1,0,0,1,0,1,1,0,0,1
mirai_06,1,0,0,0,0,0,1,1,1,0,0
mirai_07,1,0,0,1,0,0,1,1,1,0,0
mirai_08,1,0,1,0,0,0,1,1,1,0,1
mirai 09,0,0,1,1,0,0,1,1,1,1,0
```

Monitored files
(in the infected machine)
Features for each sample
(represents whether if opened a file or not)

Figure 6: Example of features used for hierarchical clustering.

cessed with "`open`". For each file opened in the infected machine, the table shows the number of samples which have accessed those. This table shows the trend that Mirai variants tend to access various files for the `watchdog`. On the other hand, as for a file `/proc/net/route` including network information, all the Gafgyt samples have accessed it. Such a trend is also seen in the majority of Bashlite samples. This result suggests that the details of files accessed by malware may be used to distinguish characteristics for each family. Next, Table 7 summarizes files that the samples accessed by invoking "`access`". From the table, it indicates that Gafgyt is more sensitive to the presence of a file "`/usr/bin/python`" in all families. In other words, this characteristic may differentiate Gafgyt from other families. Next, Table 8 describes how many of the samples in each family have posed the network behaviors specific to IoT malware. Note that these network behaviors handle network data like List 4 and List 5. As shown in Table 8, from network perspective, Bashlite tends to indicate more characteristics than Mirai and Gafgyt. Overall, this result indicates that the presence of network behavior differentiates Bashlite from other families.

Finally, we apply a hierarchical clustering method to the features we obtained through a large scale analysis. This aims to retrieve further information on the IoT malware family. As an example, we take 9 samples of Mirai and visualize their relationship as a dendrogram. For this purpose, we first focus on some files opened by these Mirai samples as features. Specifically, for each sample, we make a sequence of binary representing whether if the sample opened each file. Then, we use these sequences as features (Figure 6) and apply hierarchical clustering to make a dendrogram. To measure distances between samples, we use Jaccard distance, like some studies (Jang et al., 2011), since this is adequate in cases where binarized data are used. Eventually, as shown in Figure 7, Mirai variants can be split into multiple clusters. This result shows a possibility that our system can identify a group of samples having similar dynamic characteristics. Besides, this visualization indicates the characteristics that malware may exhibit. The figure shows that even though some malware samples are given the same label (`Trojan.Linux.MIRAI.SMMR1`, actually), they are slightly different from each other. Moreover,

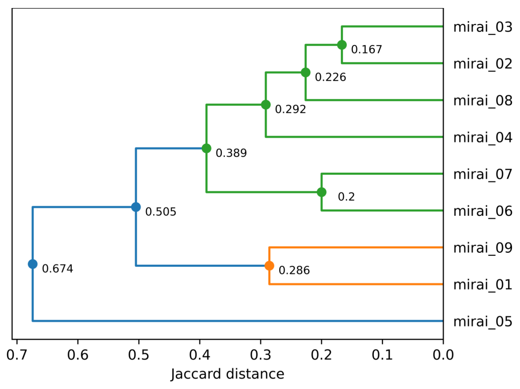


Figure 7: Example of hierarchical clustering in Mirai variants, split into three clusters.

```
0x00009528  0dc0a0e1  mov ip, sp
0x0000952c  00d82de9  push {fp, ip, lr, pc}
0x00009530  04b04ce2  sub fp, ip, 4
0x00009534  64d04de2  sub sp, sp, 0x64
0x00009538  70000be5  str r0, [var_70h]
0x0000953c  8c009fe5  ldr r0, str.https:__lmgtfy.com__q_how_to_XXX;
"https://lmgtfy.com/?q=how+to+XXX" <= Website specified by malware
author
0x00009540  f4feffeb  bl sym.imp.puts
0x00009544  70301be5  ldr r3, [var_70h]
0x00009548  003093e5  ldr r3, [r3]
0x0000954c  2d204be2  sub r2, fp, 0x2d
0x00009550  0300a0e1  mov r0, r3
0x00009554  0210a0e1  mov r1, r2
0x00009558  09ffffeb  bl sym.imp.realpath
```

Figure 8: A portion of the identified code block of Amnesia (This example uses radare2 for the display.).

this result actually supports evidence that analyzing a large number of samples allows retrieving more insights than a case only analyzing a single IoT malware binary. We expect that, by focusing on the characteristics that malware is likely to exhibit, it allows analysts to understand the severity of the threat.

4.3 A Case Study of DBI/VMI: Identify the Malware Behavior

This experiment shows advantages of DBI/VMI that Tamer can support. For this experiment, we take the analysis of Amnesia[6], an IoT malware. According to experts (Unit42, 2017), this malware has the destructive functionality to delete all files once it detects the analysis environment by checking for specific files(e.g. `/sys/class/dmi/id/sys_vendor`) .

Identifying malware behavior automatically is insufficient to traditional analysis methods. Traditional analysis methods, like monitoring system calls and network data, focus only on information about individual actions. There is a challenge that a single action is not insufficient, and it is necessary to take multiple actions into account. Therefore, we expect that such a situation requires using advanced binary techniques like DBI/VMI.

For this experiment, in order to identify malware behavior by taking multiple system actions into account, we apply dynamic taint analysis (Schwartz et al., 2010) that is based on DBI/VMI. Specifically, we hooked a invokation of system call when the monitored file(`/sys/class/dmi/id/sys_vendor`) was read, and set that data to be tracked. By tracking data flow using taint analysis, instructions related to operations on the data in the monitored file can be collected. As a result, we obtained information about instructions that are included in a basic block of the destructive function of Amnesia. This result supports evidence that, by tracking data flow, it is feasible to extract information to identify the type of malware behavior. Finally, Figure 8 shows an example of the outcome extracted by Tamer. This figure shows the code block involved in operating the monitored file.

[6]MD5: 1497740fa8920e4af6aa981a5b405937

5 DISCUSSION

5.1 Applications for Generating Dataset

Our sandbox serves functionalities that can find applications in some security fields. We describe some cases in the following: First, one example is based on a perspective of network-level detection. We assume that our sandbox can be leveraged to collect network fingerprints. As described in Section 4, our system is capable of observing some parts of network interaction between the malware and the C&C. In particular, the characteristics of the network traffic generated by malware can be applied to build a signature for detection. Next, from a perspective of host-based detection, we expect that the researcher uses our sandbox to generate a dataset about host-based behavior. For instance, as shown in Section 4, we have demonstrated that our sandbox allows retrieving a list of syscalls that the target malware invoked.

5.2 Limitations

Our sandbox system is based on QEMU emulator. This means that, if malware detects being executed inside the QEMU, it may change the behavior to hinder behavioral analysis. While the evasion malware problem is out of scope for this paper, some previous works have explored insightful knowledge (Bulazel and Yener, 2017). On the other hand, the drawbacks against VM evasion malware are an inevitable subject on debating about malware analysis methods. Therefore, we hope that our approach will also be considered as an alternative method for analyzing malware.

Besides, regarding evasion techniques of malware, there is a possibility that our system could be thwarted when malware detects PTRACE used by `strace`. As for this, we acknowledge that collecting system behavior using `strace` is straightforward. Thus, it is necessary to keep considering another technical option that could complement the disadvantages of our technical choice. For instance, eBPF (eBPF, 2020), installed in the modern Linux kernel, allows running monitoring programs in the kernel so that it may be expected as a more advanced system monitoring method than `strace`.

Moreover, as for network behavior analysis by Tamer, we acknowledge that the analysis outcomes for network behavior (like shown in Section 4.3) are

Table 9: The differences of functionalities between Tamer and Linux-based sandboxes.

	Support ARM	Support Multi-architecture	Support C&C listener	Support DBI/VMI
Limon(Monnappa, 2015)	No	No	Yes	No
Cuckoo(Cuckoo, 2013)	No	No	No	No
IoTBOX(Pa et al., 2016)	Yes	Yes	No	No
V-Sandbox(Le and Ngo, 2020)	Yes	Yes	Yes	No
Tamer	Yes	No	Yes	Yes

only a part of the network data of the malware and the current design of our fake network is simple. Thus, to elicit deeper conversation between malware and C&C server, it could require more elaborate efforts, like the protocol reversing to generate the exact command data accepted by the target malware.

5.3 Comparison with Similar Studies

We describe the differences between Tamer and existing studies. First of all, while the sandbox for IoT malware has received less attention, we witnessed some studies that handled the analysis environment for IoT malware. Pa et al. (Pa et al., 2016) have proposed IoT sandbox (IoTBOX) which analyzes the Telnet-based attacks against various IoT devices running on various CPU architectures including x86, ARM, MIPS, and so on. Their work mainly aims for analyzing network behavior, however, our aim is on analyzing IoT malware characteristics from perspectives of network-level and host-level. Le et al. have proposed V-Sandbox (Le and Ngo, 2020) for dynamic analysis of IoT botnet. Besides, V-Sandbox supports analyzing binaries for various CPU architecture, and provides C&C server simulator to support the ability to make C&C connections. We acknowledge that, different from IoTBOX and V-Sandbox, our system does not support multi-architecture CPUs. However, Tamer combines advanced binary analysis techniques such as DBI and VMI that are not supported by previous works.

Further, though they are not necessarily referred to as sandboxes, related to dynamic analysis for IoT systems, Avatar (Zaddach et al., 2014) and its successor Avatar2 (Muench et al., 2018) are sophisticated frameworks to tackle complex problems in dynamic analysis for embedded system firmware. In particular, Avatar(s) act as a software proxy between QEMU and the physical hardware in order to allow analyzing embedded system's behavior involving operation to the physical device. But, we assume that Avatar(s) are not necessarily suited for automated analysis since they require to be configured for situations depending on the physical devices.

Overall, Tamer serves another perspective from existing studies. We acknowledge that this study benefits from existing technologies, but we have assembled these into a unique combination. Thus, we assume that Tamer may act as another option for IoT malware analysis. Furthermore, in the current prototype, Tamer supports IoT malware analysis only for the ARM. But, we believe that our approach based on the auto-manipulation using `expect` can be generalized to other architectures, like we conduct analysis with `strace` in Section 4.3. Finally, a summary of the diffrerences between Tamer and existing Linux-based sandboxes are shown in Table 9.

6 CONCLUSION

This paper presents a sandbox system for analyzing Linux malware samples that infect IoT devices. Our sandbox, called Tamer, has some features to automate and facilitate IoT malware analysis, like the automated interaction mechanism using the expect utility and the fake network environment that we carefully designed. Moreover, Tamer adopts features like DBI and VMI in a unique combination, which may allow more advanced analysis. We demonstrated that our sandbox system can analyze IoT malware (e.g. IoTReaper, Amnesia) that are designed to infect IoT devices. Through a large scale analysis, we have demonstrated that our system can analyze a large amount of IoT malware samples in an automated manner, and highlight recent trends in IoT malware. From the analysis result, we have suggested a possibility that our system can identify a group of samples having similar dynamic characteristics. We acknowledge that further development and evaluation is needed to support our claim. In our future work, we will extend performing analysis on a large volume of malware and focus on extracting useful information for malware countermeasure in more detail. We hope that our study fosters discussion about the methodology to understand the dynamic characteristics of IoT malware.

REFERENCES

Bayer, U., Moser, A., Kruegel, C., and Kirda, E. (2006). Dynamic analysis of malicious code. *Journal in Computer Virology*, 2(1):67–77.

Bulazel, A. and Yener, B. (2017). A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web. In *Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium*, pages 1–21.

Carrillo-Mondéjar, J., Martínez, J., and Suarez-Tangil, G. (2020). Characterizing linux-based malware: Findings and recent trends. *Future Generation Computer Systems*, 110:267–281.

Cozzi, E., Graziano, M., Fratantonio, Y., and Balzarotti, D. (2018). Understanding linux malware. In *2018 IEEE symposium on security and privacy (SP)*, pages 161–175. IEEE.

Cozzi, E., Vervier, P.-A., Dell'Amico, M., Shen, Y., Bilge, L., and Balzarotti, D. (2020). The tangled genealogy of iot malware. In *Annual Computer Security Applications Conference*, pages 1–16.

Cuckoo (2013). Automated Malware Analysis. https://www.cuckoosandbox.org/.

Debian.org (2014). "Debian Squeeze and Wheezy armel images for QEMU". https://people.debian.org/~aurel32/qemu/armel/.

Dolan-Gavitt, B., Hodosh, J., Hulin, P., Leek, T., and Whelan, R. (2015). Repeatable reverse engineering with panda. In *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*, page 4. ACM.

eBPF (2020). "eBPF - Introduction, Tutorials & Community Resources". https://ebpf.io/.

FortiGuard (2017). Reaper: The Next Evolution of IoT Botnets. https://www.fortinet.com/blog/threat-research/reaper-the-next-evolution-of-iot-botnets.

inetsim (2020). "INetSim: Internet Services Simulation Suite". https://www.inetsim.org/.

Jang, J., Brumley, D., and Venkataraman, S. (2011). Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 309–320.

Le, H.-V. and Ngo, Q.-D. (2020). V-sandbox for dynamic analysis iot botnet. *IEEE Access*, 8:145768–145786.

LinuxFoundation (2020). "networking:bridge [Wiki]". https://wiki.linuxfoundation.org/networking/bridge.

Michel Oosterhof (2014). "Cowrie SSH/Telnet Honeypot". https://github.com/cowrie/cowrie.

Monnappa, K. (2015). Automating linux malware analysis using limon sandbox. *Black Hat Europe 2015*.

Muench, M., Nisi, D., Francillon, A., and Balzarotti, D. (2018). Avatar2: A multi-target orchestration platform. In *Proc. Workshop Binary Anal. Res.(Colocated NDSS Symp.)*, volume 18, pages 1–11.

Pa, Y. M. P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., and Rossow, C. (2016). Iotpot: A novel honeypot for revealing current iot threats. *Journal of Information Processing*, 24(3):522–533.

Schwartz, E. J., Avgerinos, T., and Brumley, D. (2010). All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Security and privacy (SP), 2010 IEEE symposium on*, pages 317–331. IEEE.

Unit42, P. A. N. (2017). New IoT/Linux Malware Targets DVRs, Forms Botnet. https://unit42.paloaltonetworks.com/unit42-new-iotlinux-malware-targets-dvrs-forms-botnet/.

VirusShare (2020). "VirusShare". https://virusshare.com/.

Willems, C., Holz, T., and Freiling, F. (2007). Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, 5(2):32–39.

Zaddach, J., Bruno, L., Francillon, A., Balzarotti, D., et al. (2014). Avatar: A framework to support dynamic security analysis of embedded systems' firmwares. In *NDSS*, volume 23, pages 1–16.