RIoTMAN: A Systematic Analysis of IoT Malware Behavior

Ahmad Darki, Michalis Faloutsos
adark001@ucr.edu, michalis@cs.ucr.edu
University of California Riverside

ABSTRACT

How can we conduct dynamic analysis on IoT malware efficiently? A key challenge is that such malware target a plethora of different devices, which makes identifying the target device non-trivial. This problem does not appear nearly as much in PC and smartphones malware, where the devices are more uniform. The contribution of our work is two fold: (a) we develop RIoTMAN, a comprehensive emulation and dynamic analysis approach, and (b) we study the network behavior of 3024 IoT malware systematically. The power of our approach lies in two key novelties: (a) Iterative Adaptation, and (b) Automated Engagement. First, we employ an intelligent iterative process that incrementally “builds” the configuration of the target device. Second, our platform automates the interaction with the malware even during the C&C server communication phase. In our experiments, we first show that we achieve an activation rate of 93% for our binaries, including 173 binaries, which Virustotal fails to identify as malicious. Second, we impersonate the C&C server for 79% of the malware binaries successfully: we make the malware initiate DDoS attacks, or enter its proliferation phase. Finally, we observe several interesting malware techniques, including unusual communication behaviors. Our goal is to release our platform as an open-source tool to accelerate the efforts for understanding IoT malware in depth and at scale.

ACM Reference Format:

Ahmad Darki, Michalis Faloutsos. 2020. RIoTMAN: A Systematic Analysis of IoT Malware Behavior. In *The 16th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '20)*, December 1–4, 2020, Barcelona, Spain. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3386367.3431317>

1 INTRODUCTION

IoT malware has emerged as a significant security threat, while all indications suggest that the problem will become more acute. First, IoT malware enabled the most intense DDoS attacks to date [10]. Second, there is widely-available source code of IoT malware, such as *Mirai* [9], *Lightaidra*, and *BASHLITE* [8], making it easy for black hat hackers to create botnets of infected IoT devices. Third, such malware employ capabilities to target particular types of IoT devices and perform different malicious operations such as harvesting network traffic in routers as it has been observed by *VPNFilter*

Figure 1: Our automated iterative approach is able to activate and impersonate the C&C server for an IoT malware, as a variant of *Linux.Tsunami*: a) Runs 1-3: the malware terminates, b) Run 4: the malware activates and sends its first packet, c) Run 5: the malware connects to our “impersonated” C&C server, d) Run 6: the malware responds to commands from our C&C server.

APT [63]. Fourth, there is evidence of IoT malware getting better: new families appear and existing families evolve and adopt sophisticated techniques [10, 66].

Securing IoT devices requires new and specialized security approaches, as solutions for computers and smartphones have limited applicability [18, 22, 31, 35, 72]. The key difference is that the term IoT encompasses a vast number of different types of devices, including industrial controllers, home sensors, and the power-grid infrastructure. Market analysis expects 5.8 billion Internet-connected IoT devices by the end of 2020 [33]. As a result, malware is often highly-specialized targeting small number of types of devices. We elaborate on the challenges of combating IoT malware in Section 2.1 and 6.

How can we profile the communication behavior of an IoT malware binary? This is the problem we address in this work. We focus on conducting dynamic analysis, since static analysis can typically give limited results, especially for packed and obfuscated binaries. Specifically, given an IoT malware binary, the goal is to: (a) activate the malware, by providing the appropriate target device configuration, and (b) engage in communication with the malware in order to explore as much of its communication behavior as possible. We make the following realistic assumption: we have no prior knowledge about the malware binaries. This makes the problem harder, but also more relevant in practice. In addition, the focus of this work is Linux-based IoT malware.

Challenge 1: Activating the malware. We say the malware is *activated*, when the malware completes its infection phase, and attempts to establish communication with the outside world. The challenge is to identify and establish the right target device for

This work is licensed under a Creative Commons Attribution International 4.0 License.

CoNEXT '20, December 1–4, 2020, Barcelona, Spain
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7948-9/20/12.
<https://doi.org/10.1145/3386367.3431317>

each binary. Each malware may expect device-specific files, such as drivers and libraries, from a large number of device configurations. In the absence of the requested files, the binary terminates its execution.

Challenge 2: Minimizing the manual effort. Reducing the manual effort is critical for studying malware behavior effectively and at scale. This includes both activating the malware, and attempting to explore the different behavior across the phases of its life-cycle. Without automated tools, both these tasks require trial and error attempts and significant human effort.

To the best of our knowledge, there are no studies that focus on profiling the network behavior of IoT malware binaries at scale. For example, the most relevant effort to our work [24] develops a dynamic analysis tool, but it does not analyze the communication behavior of the malware and focuses mainly on profiling it at the OS level. In Section 7, we briefly review previous efforts in the following categories: (a) sandboxes and emulators [27, 34, 47, 59], (b) studies of malware traffic in the wild [10, 17, 46], (c) IoT firmware analysis [18, 23], and (d) static and dynamic analysis of malware [20, 21, 40, 43].

As our key contribution, we develop **RIoTMAN** (Riverside IoT Malware Analysis.), a comprehensive emulation and dynamic analysis platform for Linux-based IoT malware. Our approach is designed to address the two aforementioned challenges by employing two complementary techniques: (a) **Iterative Adaptation**, and (b) **Automated Engagement**. We depict the operation of our approach in Figure 1. After several failed attempts, we create the required target configuration that makes the malware activate in the 4th iteration of execution. After that point, we begin to “control” the malware by impersonating its Command and Control (C&C) server. We revisit this example in Section 4.

Novelty 1: Iterative Adaptation. For activating the malware, we introduce the Iterative Adaptation process, which incrementally builds the desired device configuration. We start with a basic hardware configuration, and keep executing the malware, while fine-tuning the configuration.

Novelty 2: Automated Engagement. We introduce a suit of automation techniques that minimize the need for manual effort. We collectively refer to these techniques as the Automated Engagement process. This automation facilitates two distinct phases: (a) generating the target device configuration, and (b) impersonating its C&C server.

The key advantages of our approach are: (a) **cost effectiveness**: we have one “evolving” emulation instead of a large array of fixed device configurations; and (b) **wide-behavior exploration**: our method enables us to engage with the malware and observe its multi-faceted behavior.

We evaluate the effectiveness of our approach using 3024 malware binaries, from public and private sources (details in Section 2). We highlight our key results below.

1. Activation success: 93%. We show that the Iterative Adaptation works: we activate 93% of our binaries by incrementally refining our initial base configuration. Interestingly, we manage to activate all our packed and obfuscated binaries. In achieving this, our approach constructs more than 700 unique device configurations.

2. Activating 173 binaries that Virustotal misses. We are able to activate and observe malicious behaviors from 173 binaries, which Virustotal [65] classifies as benign. We also identify the main malware behaviors that seem to evade or confuse other approaches in Table 3.

3. C&C impersonation success: 79%. We impersonate the C&C server for 79% of all our binaries automatically. We consider the impersonation successful, if at least one command issued will make the malware initiate a new behavior, such as a DoS attack, or start proliferation phase. We manage to engage with binaries from more than 20 major IoT malware families.

4. Identifying 118 binaries with uncommon communication behaviors. We identify binaries that do not attempt a C&C connection, but instead: a) operate in a peer-to-peer fashion, b) passively wait for a connection, or c) eschew all communication and, instead, “brick” the device.

5. Observing “cross-talk” across families for 294 binaries. We find that 294 binaries use a C&C communication protocol from a different IoT malware family than the one they belong to. Upon closer investigation, we find that some of these malware are hybrids and combine source code from different families, as we discuss further in Section 5.

Tangible outcomes. Our work provides the following artifacts to further the research in this area:

a. An open-source platform: We release our platform as an open-source tool to leverage the collective power of the community (link: <https://github.com/darkerlord/RIoTMAN2020>).

b. Communication traces: We will provide extensive packet traces of the network layer behavior of our binaries. These packet traces currently consist of 20 minute executions for an aggregate of more than 2 TBs of packet traces. We also provide longer execution traces for select binaries for longitudinal behavior studies.

We argue that a concentrated community-backed effort leveraging RIoTMAN could significantly enhance our ability to understand and mitigate IoT malware in depth and at scale by providing: (a) behavioral signatures, and (b) intelligence for detecting and containing the malware.

2 CHALLENGES AND DATA

We discuss the challenges in analyzing IoT malware and we present our malware datasets.

a. Basic concepts and terminology. The term “Internet of Things” (IoT) device refers to any computational interconnected device, excluding traditional computers and smartphones. IoT devices have lower computation power, and are based on different CPU architecture from PCs, and therefore different security measures is required [30, 38]. Here, we focus on Linux-based IoT devices on embedded CPU architectures, such as ARM or MIPS.

b. Malware activation challenges. The first major challenge in analyzing IoT malware is to emulate its target device. In PC and smartphone research, emulating a target device is relatively standardized and well established. However, the IoT devices vary significantly [18, 72], and different malware may target a subset of such devices. There are two types of resources and configuration requirements that need to be met for a successful emulation:

Dataset	DMain	DNew
No. of Binaries	2885	139
Confirmed by Virustotal	2711	139
Missed by Virustotal	173	-

Table 1: Our two dataset DMain and DNew and their assessment form Virustotal. Virustotal did not classify 173 binaries as malicious.

1) Software resources: This type of resource consists of configuration, binary, and executable files, which the malware attempts to find within the emulation configurations. Different types of requested files need to be handled differently. For example, a dynamically linked library is observed in the ELF structure of the binary, while a request for configuration file is observed at the OS level.

2) Hardware resources: This is a more challenging resource request and resolving it requires emulating the actual hardware peripheral. As an example, a malware can request configuration information from the non-volatile memory (NVRAM), which we discuss further in Section 3.4.

c. Communication and engagement challenges. Our goal is to engage with the malware and make it reveal a range of its behaviors. To achieve this, the emulator should imitate both the rest of the botnet, and the Internet, e.g. DNS servers, vulnerable devices.

1) Identifying the type of communication: Each binary may initiate connections with different purposes, such as connecting with its C&C server, checking for Internet availability, or scanning the network to proliferate. If we can differentiate between these connection types, we can create suitable responses more effectively.

2) Effective Internet “impersonation”: The goal is to create a contained environment with connection availability for the malware without access to external network to avoid further malware proliferation. This includes handling DHCP or DNS requests as well as server impersonation. This requires us to respond properly to each communication requests initiated by the malware.

2.1 Our malware data

We use two malware datasets and a source code archive.

A. Our large malware dataset: DMain. At the time of writing this work, there were no public or easily accessible repositories of IoT malware. We were able to collect **2885** malware binaries from July 2014 to January 2018 using two types of sources:

a. Honeypot platforms: We acquired binaries from multiple honeypot sensors located in widely-spread geo-locations. The honeypot sensors listen on commonly used ports related to Telnet, SSH, UPnP, and other ports that have been reportedly used in IoT attacks [67].

b. Forums and Blogs: We collected an additional 350 malware binaries from security researchers that specialize in IoT malware analysis to whom we are grateful.

In this dataset, there are 173 malware binaries that are deemed as benign by all the detection engines in Virustotal. However, these binaries are malicious and exhibit botnet behavior when observed by RIoTMAN, as we discuss in Section 4.

Figure 2: The architecture of RIoTMAN: six modules that activate and engage with malware binaries using: (a) Iterative Adaptation, and (b) Automated Engagement.

B. Our newer malware dataset: DNew. We also use a newer dataset to further test our approach. These binaries were collected after December 2018 and were not part of our initial malware set. These datasets were created from two sources: (a) Virustotal: a set of 67 malware binaries collected via Virustotal academic API [65], and (b) Honeypots: we have collected 72 malware binaries from our contacts within several Honeypot projects [69].

Our datasets encompass at least 20 malware families. Our binaries span more than 20 malware families using the malware classification provided by Virustotal [65] as we explain below. Most importantly, our dataset has representative binaries from all the malware families reported in F-Secure’s “2019 IoT threat landscape” [28], arguably one of the most recent and complete reports. Note that many families encompass distinct variants, which are referred to as sub-families.

Identifying families and sub-families using Virustotal. Virustotal is a widely-used reference point service for malware analysis [65]. For a submitted binary, it returns the results from more than 50 detection engines. We perform a “majority vote” on the suggested family from these results. Although the accuracy of Virustotal is not perfect [54], we find this approach sufficient for estimating overall coverage and trends here. We use and discuss this classification in Section 5.3.

C. Malware source-code dataset. Apart from malware binaries, we were able to collect 174 source code of IoT malware from public software repositories. Specifically, we found malware source code on public websites, such as Github[58] and Pastebin, often with instructions on how to create a botnet. We use the source code to help us automate the C&C impersonation capability, as we discuss in Section 3.3.

3 THE RIOTMAN ARCHITECTURE

The novelty of RIoTMAN is two fold: (a) Iterative Adaptation, and (b) Automated Engagement. These capabilities work in a synergistic way to: (a) incrementally create the malware’s target device configuration starting from a basic setup, and (b) minimize the need for manual effort. The system consists of several modules, which complement each other. We depict the system architecture in Figure 2 and we present an overview of the role and interactions of each module below.

3.1 The Supervisor module

This module oversees the whole operation. It interacts with the user, initiates the analysis of a binary, and coordinates the execution of the other modules, until the analysis is completed. It drives the Iterative Adaptation process, which is a key novelty of our approach.

Its key functions include: (a) collecting information from the Static Analysis module, (b) initiating the emulation at the Sandbox module, (c) interpreting the information from the Profiler module, and (d) repeating the dynamic analysis to extract as much information about the malware as possible. Each iteration is a dynamic execution of a given malware followed by analyzing the execution traces for further iterations. This module orchestrates the series of analysis iterations by taking into account if the malware has successfully activated and was successfully engaged in communication. Otherwise it will continue enhancing the Sandbox and the NetComm setup using the information derived from Profiler. The analysis cutoff is 12 iterations as a practical constraint to ensure that the analysis does not continue for an arbitrary amount of time.

3.2 Static Analysis module

This module performs static analysis on any new binary in order to extract information, which could guide the execution. The information can include the target CPU architecture and dynamic linked libraries. When the binary is neither packed nor obfuscated more information can be extracted. This module performs two types of analysis on a given binary:

1. ELF analysis: By using `readelf` [32] we analyze different sections of each ELF file and identify used CPU architecture, and information about dynamically linked libraries. RIOTMAN uses this data to select the emulation's CPU architecture and dynamically linked libraries.

2. Structural analysis: If the binary is not using runtime packers or obfuscation, it is possible to extract more information via static analysis. Although in this work we focus on dynamic analysis as the main analysis technique, this module extracts artifacts such as channel names and passphrases in IRC based botnets. In the current implementation, we created IDAPython scripts compatible with IDA Pro [37] Python API to extract such information. However recent work [60] discovered that disassemblers are less effective in analyzing malware binaries targeting less common CPU such as architectures ARM and MIPS.

3.3 The Automation module

One of the key novelties of RIOTMAN is the Automation module; it facilitates the Iterative Adaptation operation for the Supervisor and enables the Automated Engagement.

A. Supporting the Iterative Adaptation. The Automation module attempts to provide the configuration requirements for each iterative execution. To automate this, we use a Configuration database of device configurations of IoT devices. We bootstrapped our database with the firmware from 10 often-targeted IoT devices: six routers (Linksys, Netgear, TPLink), a DVR device, a NAS device, and two Ubiquiti Network devices. Interestingly, some binaries request files from multiple IoT firmware: in other words, our approach re-creates device configurations that have not been explicitly

provided. Below, we show how this module handles each type of resource requests.

1. Software resources: Based on the execution constraints of the malware, a copy of the software resource such as library and configuration files from the Configuration database are placed in the requested directory in the Sandbox. In case the malware binary is dynamically linked, the appropriate standard library with the compatible kernel is used.

2. Hardware resources: When the malware fails to access hardware peripherals for reading and/or writing data, a user space replica of the hardware resource will be added to the Sandbox. More details are discussed in Section 3.4.

In case when the malware requests file that is not available in our Configuration database, RIOTMAN creates a "dummy" file that replicates the type of file (e.g. binary or text) in response to the requested resource. This approach often satisfies the malware request as we see in Section 4.1. The same technique is employed in previous efforts which also resulted in effective solution [18, 72].

B. Automating the C&C Impersonation. After activating the malware binary, we aim to impersonate its C&C server for it. The goal is to correctly identify: (a) the communication end point that malware receives commands from, and (b) the communication protocol in use. As discussed in Section 2.1 we collected 174 IoT malware source codes, which we synthesized and distilled to create a library of C&C server models to incorporate in the NetComm module. Upon malware activation and enabling network connectivity, RIOTMAN sinkholes all communication requests initiated from the malware using the NetComm module. The Automation module selects the appropriate C&C server communication protocol through iterations of executions. Following highlights our approach:

a. Select C&C protocol based on artifacts: The Profiler module analyzes the packets collected via the sinkhole server, and extracts revealing keywords. For example, if the packets include keywords related to IRC such as NICK, we use the IRC C&C protocol. Along with extracting information from the packets, we also use the information collected from the Static Analysis module as well, which includes IRC channel name as well as passphrase to join the channel. We have created a dictionary of keywords that are indicative of different types of C&C protocols which are collected by examining the malware source codes in our repository (Section 2.1).

b. Try all available protocols: If extracting information from the packet is not effective, we exhaustively iterate through all the possible C&C protocols and select the one that will direct the malware to engage in malicious activity. Otherwise, no C&C impersonation is possible for the given malware.

3.4 The Sandbox module

This module creates a fully functional emulated IoT device based on the configuration instruction that it receives from the Supervisor module. In each execution, the Sandbox creates an emulation instance through the following steps:

1. Device emulation build on top of QEMU. We use QEMU [12], an open source project, for CPU emulation. We modified QEMU substantially by adding/modifying ~2000 lines of code (LoC). The goal was to introduce the networking and monitoring capabilities

Figure 3: Building on top of QEMU (~2000 LoC): Enabling parallel executions of multiple emulation instances, while providing isolation at the network level using a bridge. The NetComm module emulates the Internet and drives the Automated Engagement.

required to engage in communication with the NetComm as we explain later.

A key engineering contribution is that we enable the execution of multiple instances in parallel with network isolation. To achieve this, we extend QEMU to have one or many tap software device [68] and communicate with the rest of RIoTMAN modules through the bridge interface in the experimental host machine. This is a crucial setup to send/receive network traffic to/from QEMU instances. More specifically, the traffic observed from QEMU is the Data Link layer traffic with the instances' Ethernet MAC address as the source or destination. Our goal is to intercept this traffic in the NetComm module as we explain in Section 3.5. Figure 3 shows several emulation instances that are connected to a network bridge to keep the networking operations of each instances isolated. Through the bridge, each instance obtains its IP address from the DHCP service and connect with the NetComm module which simulates the rest of the Internet.

2. Establishing a filesystem and configuration resources. The required filesystem consists of the files and binaries that are recommended for each new execution by the Supervisor module. A configuration resource can correspond to: (a) vendor specific library files e.g. libnat, libacos_shared that are available in *Netgear* home router firmware, (b) shell binaries that are used to modify IoT devices configurations, such as *sncfg* in *D-Link* routers or *cfgmtd* in *Ubiquiti Networks* devices, and (c) device configuration files, such as */mnt/Config/* in *ZyXel* devices.

3. Establishing the Linux kernel base. We use a Linux kernel based on OpenWrt [2]. By default, we use the glibc C library and, depending on the resource requirements, the Sandbox can replace it with a different C library, such as musl or uClibc. Furthermore, we modify the kernel to support: (a) system call monitoring, and (b) hardware peripheral emulation, which we discuss below.

a) **System call monitoring:** We monitor the system calls in the kernel with the use of kernel probes (kprobe). We opt for kprobe over user-space monitoring tools, such as strace, in order to avoid being detected by malware. We see the importance of this approach when malware utilizes evasion techniques in Section 5.1.

b) **Hardware peripheral emulation:** Given the wide range of target devices, there is a large number of peripherals that the malware may request to access. The goal is to intercept such system calls and respond in a way that will satisfy the malware. This is well-known hard problem [35, 71]. We highlight a particular type of request to show the complexity of making the emulation work. Considering requests to the non-volatile memory (NVRAM); we created a basic libnvram, so that works in a key-value fashion based on the requests made to NVRAM. As we show in Section 4, this approach has been effective in allowing malware to access a simulated NVRAM and start executing.

3.5 The NetComm module

The NetComm module provides network communications for the Sandbox which includes network services such as DHCP and DNS services as well as enabling network interaction with the malware to explore its network layer behavior. The Sandbox instances send communication traffic to this module through the bridge device, described in Section 3.4, which is then received by the NetComm module via iptables by creating redirecting rules into the PREROUTING chain. This approach is effective in engaging in communication with the Sandbox and the understudy malware. In addition, this will allow isolating RIoTMAN from any external network connectivity and stopping further proliferation of the malware. A malware may initiate a network connection for different purposes. The communication functionality can be grouped into three main types:

a) **Providing Internet access.** The goal is to provide network availability to make the malware believe it has access to the Internet. This setup works based on the information derived from previous iterations to identify the target device and the services: (a) identify the protocol of the contacted endpoint service, (b) create a process that impersonates the desired network service, and (c) direct all the relevant malware traffic to it.

b) **Impersonating the C&C Server.** The goal here is to impersonate¹ the C&C server of the malware using the Automation module, which we described in Section 3.3. The C&C impersonation is done via a list of pre-defined commands, from the Automation module, that are sent to the malware. Each command is confirmed to be a successful one if: (1) the C&C connection is maintained, and (2) the malware starts generating new traffic or exhibit new system level behavior. This helps us reveal interesting malware behaviors: we can make the malware enter its proliferation phase or terminate itself, and we are successful in doing so, as we see in Section 5. In the current implementation we do not consider impersonating encrypted C&C communication which will remain as a future work.

c) **Intercepting malware proliferation.** The goal is to identify the proliferation techniques that malware uses. The most common proliferation approach is to start scanning a wide range of IP addresses in one or multiple port numbers. We identify this behavior by the increase in the number of connection requests and the number and the type of packets. In our current implementation, we use Cowrie [53] for SSH, Telnet scanners, and a simple HTTP server

¹Our goal here is not to reverse engineer the communication protocol, which is a research topic in its own right [15, 16, 25].

for attacks targeting web UI and *UPnP* interfaces. The honeypot instances are created based on port numbers: for example, to capture the proliferation via the *UPnP* service the module starts a server listening for TCP connections on port 7457.

In the current implementation, RiOTMAN only supports IPv4 connection. At the time of writing, there has been no IoT malware observed that uses IPv6 for communication.

3.6 The Profiler module

The Profiler module creates a detailed profile of the behavior of each malware binary by analyzing each execution. A key novelty here is that: (a) we monitor the malware at the OS, and network level, and (b) we combine the information to understand the intent of each malware action.

A. OS level analysis: The module analyzes the system call trace of the whole emulated system with a focus on malware specific generated system call traces. In this work, we use data dependency analysis [40] on the traces. We convert the execution trace to a dependency graph, where nodes show the system calls, and the edges indicate the connection between system calls. E.g in a trace where a file is opened to read from, the dependency graph will have 3 nodes, open, read, and close and 2 edges showing that first the file was opened, then was read from, and later was closed. This technique hides subtleties and challenges (more than 20k line of Python code). Using this technique, we can identify the root cause of a failure in the malware execution. We analyze the malware behavior based on the group of system calls:

a. File access system calls: We analyze the execution trace focusing on file access system calls, such as open and fstat. By examining the required permission (read/write) for the resource and the return value, we identify missing files.

b. Network system calls: We monitor network system calls, such as connect and sendto, to identify the target server and the type of connection, as we explain in Section 3.5. Note that some network system calls reveal activities that do not generate network traffic: for example, some binaries create packets for a non-existent network interface, or create raw sockets to sniff network traffic.

B. Network traffic analysis: We collect all the packets using QEMU functionality, but the challenge is to identify: (a) the packets that are generated by the malware, or its child processes, and (b) the communication intent of the packet, as we discussed earlier. For the first challenge, we rely on a careful: (a) matching of network traffic with processes, and (b) understanding of process hierarchy which shows child and parent processes.

Identifying the intent of each packet. The goal here is to identify the intent and the type of communication that each packet relates to. The key challenge is to differentiate between the packets from the proliferation and the C&C discovery phase. We identify the C&C server communication based on the number of attempts that the malware makes to contact an IP address, as well as the duration of connection maintaining. Typically, the higher the number of attempts and the longer the communication time, the more likely that the destination is a C&C server [67].

4 EVALUATING RIOTMAN

We evaluate the effectiveness of our approach using the malware datasets described in Section 2.1.

A. The experimental platform: Each analysis starts with a basic configuration, which is based on a basic version of OpenWrt [2] as the embedded OS with *glibc* running within QEMU [12]. Upon malware activation, RiOTMAN engages with the malware provides network communications to explore its behavior. We highlight some implementation details below.

a. Clean starts: Each execution starts with no prior infection in the Sandbox.

b. The infection process: By default, the malware binary is placed in the */root/* directory and it is executed as part of the start up process (*init*) with root privilege. Although one could start the malware in user mode, our goal is to: (a) execute the malware without having access constraints, and (b) most of IoT devices have only root-level users.

c. Experiment duration: Each dynamic execution lasts 20 minutes. However, this can be adjusted as needed. For example, some malware employs delays and scheduling techniques to specifically evade a dynamic analysis like ours.

d. Network isolation: For safety, the server hosting the analysis environment is in a private network fully isolated from the campus network and the Internet.

B. RiOTMAN in action: a case study. We describe the iterative steps for a variant of *Linux.Tsunami* malware shown in Figure 1. We plot the number of system calls and connection attempts of the infected device for each iteration as an indication of the activity of the malware.

Iteration 1: The malware fails to activate due to the absence of libraries, including *uClibc* and *libnvr*.

Iteration 2: The sandbox provides the library files, but the malware aborts, as it fails to access the NVRAM memory.

Iteration 3: A key-value NVRAM service is enabled in the sandbox. However, the malware terminates due to the absence of a source configuration information (a device name in this case) from the simulated NVRAM.

Iteration 4: The target device is emulated successfully, and we say that the malware is activated.

Iteration 5: The activated malware establishes connection with the simulated C&C server and awaits commands. We infer the communication protocol from hints within the packet sent by the malware and with the help of our database of C&C behaviors.

Iteration 6: We begin issuing control commands, attempting to impersonate the C&C server. We successfully send a *scan* command, which makes the malware enter its proliferation phase.

4.1 Activation effectiveness of RiOTMAN

Dataset	Tot.	Activated
DMain	2885	2688 93%
DNew	139	129 93%

Table 2: RiOTMAN is able to activate 93% of the malware binaries from our two datasets. The failed 207 binaries are usually malformed or highly-specialized binaries.

Here, we evaluate the ability of RIoTMAN in activating IoT malware binaries.

Key Result 1: We activate 93% of the malware binaries. Our experiments demonstrate that our iterative approach works well: it is feasible to start with a basic configuration and morph it into the desired target device. We activate 93% of the binaries for both datasets, as we show in Table 2: we activate 2688 of the 2885 binaries for DMain and 129 of the 139 binaries from DNew. Note that we also activate all of the packed and obfuscated binaries in our database. This is a strong indication that our approach does not rely on its static analysis module in any significant way.

We investigate the 207 binaries in our datasets which did not activate, and we identify the following two main reasons:

a. **Malformed binaries:** We find that 205 binaries fail to execute due to segmentation faults and bus errors. We investigate the errors manually and find that they are due to misconfigurations in the malware, possibly in its source code.

b. **Highly-specialized binaries:** We find two binaries that target hardware resources from specific devices. For example, one of the binaries attempts to access resources from a hardware peripheral specific to *NETGEAR* home routers. The failure is due to a set of configuration values that are initialized in the boot up phase of the *NETGEAR* device, which are not straightforward to replicate without access to the hardware.

RIoTMAN compared with other approaches. We compare RIoTMAN with existing methods and tools.

a. **Comparing with Virustotal.** Virustotal is a *de facto* reference point for malware analysis as it has adopted a generous open policy.

Key Result 2: Activating 173 binaries that Virustotal misses. We are able to activate and observe malicious behavior from 173 binaries, which Virustotal reports as benign. We argue that if Virustotal was able to activate these binaries, it would have easily observed the malicious behavior, as we did. In fact, RIoTMAN goes one step further: it impersonates the C&C server for 99 (~50%) of the binaries and invokes a DoS attack for some of them.

b. **Comparing with Detux:** Detux [27] performs static analysis on the binary and uses pre-built desktop Debian instances based on QEMU to activate the malware. Detux can only activate 68% of the binaries, which is significantly lower compared to our rate.

c. **Comparing with Padawan:** Following a different approach from ours, Padawan [24] analyzes malware by using a number of pre-configured sandbox instances. We used a sample of 30 malware binaries (randomly selected from each malware family) on Padawan’s online service [4], and the results suggest an activation rate of ~85%. We will conduct an extensive comparison in the future, as their online service imposes some limitations.

Note that Padawan does not claim to have an automated network capability, and it focuses on analyzing the OS level behavior. In addition, it does not address the different sets of resources that a given malware may access during its execution which hampers with its behavior. In some cases the malware may be activated, however, it remains dormant and does not exhibit any malicious activities. Such resources range from software and hardware components as well as network availability. By contrast, RIoTMAN’s Iterative Adaptation approach makes IoT malware analysis more scalable, since it creates the emulation instances according to malware’s

activation requirements. In our evaluation more than 700 unique emulation configurations were created.

RIoTMAN’s approach in activating more malware. Diving deeper, we identify the advance malware behaviors that make activation difficult for the other methods. By examining the harder to activate binaries, we identify the 7 most common behaviors in Table 3, which we refer to as **Elusive Behaviors**. We highlight some of these behaviors below.

a. **Accessing NVRAM:** There are malware that use the configuration from NVRAM to identify the type of the device (e.g. “Device Name”, “Device Model”), and report the results to its C&C server. Failure to access the NVRAM results in immediate termination.

b. **Detecting tracer tools:** Binaries often check for the existence of monitoring tools, such as strace and ptrace. We found 6 malware binaries that invoke a PTRACE system call to check if the PTRACE_TRACEME flag is set indicating an active ptrace monitoring. In that case, the malware aborts.

c. **Evading the analysis environment:** We find 10 malware binaries that check for the presence of an analysis environment. For example, three malware binaries check for the radar2 [55] analysis tool by accessing /usr/bin/r2 binary. The seven other malware binaries check for /tmp/toexec and other similar file names to detect the presence of an automatic analysis tools, such as cuckoo [34].

These Elusive Behaviors have led to early termination of malware execution in the tools that we discussed above. For example, we study the behavior of the 173 binaries, which failed to be analyzed by Virustotal and any of its detection engines. We find that 154 of these binaries exhibit at least one of the behaviors. The remaining 19 binaries use runtime packers and data obfuscation to evade static analysis. In 12 of the 19 instances, we observe the use of custom UPX packers. In the remaining 7 binaries, we see the use of obfuscation to scramble the data part of the binary, which may make tools that rely on static analysis fail.

4.2 Effectiveness of C&C impersonation

A key novelty of RIoTMAN is that it can automate the engagement with a malware by impersonating its C&C server, as we explained in Section 3.3.

Key Result 3: RIoTMAN achieves 79% C&C impersonation success. RIoTMAN is successful in impersonating the C&C server for 79% of the binaries as shown in Table 4. Our approach makes the malware follow botnet commands, using the process which we described in Section 3.5.

We also study the number of binaries that responded to different types of commands in Table 5. All malicious activity is safely contained within the emulated environment.

a. **61% of binaries respond to Configuration or Report commands.** This group of commands take care of operational logistics, such as changing the *nickname* of the bot, resetting the *spoofing* IP used in attacks, or reporting status information, such as memory usage.

b. **70% of binaries start a network attack.** All the attacks that we initiate are DoS attacks.

c. **64% of binaries enter proliferation phase.** In the proliferation phase, the bot tries to identify and infect other devices.

Malware Behavior	Without RIoTMAN	With RIoTMAN
B1: Accessing network interface	Malware fails to execute as it cannot access certain network interfaces, such as <i>hwsim0</i> and <i>wlan0</i> , on the infected device.	By using our modified version of QEMU, the requested network interfaces are added to the Sandbox module and can be accessed.
B2: Detecting tracer tools	Malware terminates as it detects userspace process tracer such as <i>ptrace</i> used in <i>strace</i> to collect process system call.	RIoTMAN uses <i>kprobe</i> , a kernel level tracer, to collect system calls traces. Malware executes as it does not detect this tracer.
B3: Evading the analysis environment	Malware terminates, if known artifacts of analysis tools, such as <i>radare</i> [55] and <i>cuckoo</i> [34] are detected.	The Iterative Adaptation identifies any indication of a sandbox and hides or removes it.
B4: Verifying embedded device	Malware stops executing as it fails to determine that it is in an embedded device. In some cases, malware uses library files, such as <i>uClibc</i> 's, to assess the type of device.	Malware activates: after one failed executions, the Automation module provides the requested file and places it in the desired directory and the malware activates.
B5: Accessing IoT device configuration files	Malware terminates: it performs a series of device configuration file requests but none of the requested files exists.	The Automation module copies the requested files from our database. The malware "verifies" the type of device and sends the config information to its C&C.
B6: Requiring dynamically linked libraries	Malware fails to execute, since it needs to dynamically link to vendor specific libraries, such as <i>libacos.so</i> , <i>libbcm.so</i> , and <i>libnat.so</i> .	The Automation installs the appropriate library; in the next iteration, the malware starts executing without issues.
B7: Accessing non-volatile memory (NVRAM)	Malware fails to access device configuration stored in device NVRAM and it terminates.	A key-val based NVRAM library is created and populated based on the malware's config requests.

Table 3: Elusive Behaviors: a list of malware techniques that lead to early terminations. RIoTMAN manages to activate such difficult malware binaries, while many existing state of the art tools fail to do so.

Total binaries	2885	
Activated	2688	93%
Engaged	2291	79%

Table 4: RIoTMAN can impersonate the C&C server for 79% of the binaries. Versions of the Gafgyt and Mirai C&C server protocol are among the most widely used in communication.

Command Type	Malware	
Configuration or Report	1750	61%
Attack	2031	70%
Scanning	1842	64%
Termination	1684	58%

Table 5: The number of binaries that acted upon different commands received from the C&C server impersonation.

d. 58% of binaries respond to termination commands. We make the bot stop an ongoing attack or kill itself. This knowledge could be helpful in botnet containment efforts.

Key Result 4: Identifying uncommon communication behavior. We investigate the binaries with which we failed to engage. First, we failed to activate 198 malware binaries, and therefore we could not even attempt to communicate with them. We investigate the remaining 397 binaries and identify the following reasons:

a. Unknown protocol: There are malware binaries that use communication protocol with end-to-end encryption and employ specific packet exchange with their server to ensure the legitimacy of the communication. For example, upon establishing a connection with a server the malware expects an authorization "key" from the server to continue with the connection, otherwise it terminates it.

b. Peer-to-peer botnet: We identify 14 binaries that exhibit peer-to-peer behavior by creating listeners on a number of UDP ports and exchange packets with what appears to be a super-node that includes keywords such as *DHT*. These binaries appear to be instances of Hajime which are explored in a recent study [36]. So far, our focus was on C&C-based malware, but we plan to extend RIoTMAN capabilities to peer-to-peer botnets as well.

c. Proxy agent malware binaries: We observe 74 malware binaries are proxy agents waiting for a connection on particular open ports. We find that 68 of them are classified as *Proxy Agent* by Virustotal. The other 6 malware belong to the *Luabot* family [57]; they setup a proxy on an infected device, which can provide HTTP and SMTP services [5].

d. Destructive malware: We identify a class of malware that exhibits a purely destructive behavior. For example, these malware corrupts the system memory, essentially "bricking" the device, like Brickerbot [1]. We also found malware that prohibits all network communication: it installs firewall rules to drop all incoming and outgoing packets.

4.3 Understanding RIoTMAN's Iterations

We study the value of our iterative approach here. Figure 4 shows how the iterative approach manages to activate and engage in communication with more malware binaries as it learns about their resource constraints and making them available.

Iterations towards activation: Before starting the dynamic analysis of any given malware binary, the Static Analysis module identifies the target CPU architecture that is compatible with the malware. Given there are different CPU architectures commonly used among IoT devices (e.g MIPS Big Endian, MIPS Little Endian, ARM, and ARM64), selecting the appropriate one for the dynamic analysis accelerates the analysis and saves iterations of dynamic

Figure 4: As the iterative execution progresses, more malware binaries are activated and are engaged in C&C communication with RIoTMAN.

executions. In the 1st iteration 68% of the binaries are activated while the rest fail due to missing environment resources. In the following iterations, these missing resources are identified by the Profiler module and are added to the Sandbox. In doing so, RIoTMAN makes use of the Automation module and its Configuration database to find and add the missing resources in the next iterations. In the 2nd and 3rd iteration, the missing resources include library and configuration files specific to IoT devices. In the later iterations, the malware that rely on network connectivity will start activating. In our dataset, we observed more than 500 malware binaries that check the network connection availability before they engage in any malicious activity. By the 6th iteration we have activated 93% of all the given malware binaries.

Iterations towards C&C engagement: In the 1st iteration there are no C&C engagement since the NetComm module is not used. RIoTMAN’s approach is to use the information gathered in the previous iteration to identify the C&C server and the used protocol for the next subsequent iterations. In the 2nd iteration, upon their activation, NetComm creates the server impersonation for the activated malware and engages in communication with them. Certain number of activated malware binaries may fail to engage in C&C communication due to reasons explained in Section 4.2. In the following iterations the number of successful C&C engagement grows with respect to previous activation growth. This continues until the 8th iteration when 79% of the binaries successfully engage in communication with RIoTMAN’s C&C impersonation.

5 A STUDY OF MALWARE BEHAVIORS

We analyze the behavior of the malware binaries focusing on their communication behavior. We focus on the DMain dataset here.

5.1 Target device configuration variability

Our approach iteratively constructs more than 700 unique device configurations in order to satisfy 981 unique resource requests, which go beyond the initial basic configuration.

a. Software Resources: In this study, RIoTMAN has handled 973 unique requests for software resources. We find that 615 of them were resolved by our Configuration database in the Automation module. The remaining 358 requests were resolved with “dummy” files, as discussed in Section 3.3 and involved: (a) files related to scanning, such as a file that holds list of credentials for bruteforcing Telnet or SSH; (b) specific configuration files related to the malware, such as an accompanying file Zopee3ve requested by an instance of linux.wifatch malware [64]; or (c) malware loading kernel module (using modprobe) hi_rtc.ko which is a module specific to HiSilicon IP camera.

b. Hardware Resource: We find 8 types of hardware requests. Some malware binaries attempt to tamper with the watchdog timer to prevent the device from rebooting. In other cases, the malware requests to modify or read the device configuration from the NVRAM directly or by using user-space processes, such as nvram and cfgmtd.

5.2 Profiling malware communication

We provide the results of our profiling of the communication behavior of the malware during: (a) server discovery, and (b) proliferation.

A. Server discovery: connecting to the C&C server. Understanding how the malware connects to its C&C server is a critical step in detecting infections, finding ways to infiltrate the bot, and ultimately, do a botnet take-over [62]. We use the term “connecting to the C&C server” to encompass several different mechanisms that a bot may use. We find that the majority of the binaries (2580 of them) attempt to connect to their botnet. We can create a taxonomy of server discovery strategies based on the following behaviors: (a) using IP addresses versus domain names as server, (b) using one or multiple servers, and (c) using a fixed versus a dynamically-generated group of servers. In Table 7, we show number of binaries that exhibit each behavior. The largest group with 2251 binaries have used a single IP address in our experiments. We also find 5 binaries that use DGA (Domain Generation Algorithm) to connect to the server. Interestingly, we find 2 instances that use a DGA technique named ranbyus [11]. Upon investigation, we find that two binaries use the ranbyus [11] DGA technique, which could help predicting the sequence of domains that they will attempt to connect to. This knowledge could enable a botnet takeover similar to the Torpig takeover [62]. We have already started to expand on these results by conducting experiments with longer duration to profile the dynamics of these behaviors. For example, in 2 of the DGA based malware we observe that the contacted domains are repeated based on the day of the month.

B. Proliferation: how malware spreads. Extracting information on how the malware proliferates could lead to mechanisms to detect its presence in a network and create firewall rules to protect against such proliferation. In this study, we were able to get 1842 binaries to enter their proliferation phase: they start scanning IP addresses at different ports. We identify two groups of scanners which proliferate by: (a) using brute force, or (b) exploiting vulnerabilities.

a. Profiling brute-force techniques: 2000 login credentials. We find 1676 malware binaries that use the brute-force technique: the malware tries to guess the login and password of the target device. As the malware attacks fake devices within our emulation,

Family from Virustotal	Impersonation Success	Gafgyt C&C		Tsunami C&C		Aidra C&C	Mirai C&C
		Prometheus	QBot	Remaiten	Capsaicin	Lightaidra	Mirai
Gafgyt (>6 sub-families)	94%	148	1296	-	2	-	5
Tsunami (>2 sub-families)	98%	4	26	43	25	-	-
Aidra (>2 sub-families)	87%	1	5	-	-	2	-
Mirai (>2 sub-families)	86%	-	-	-	-	-	402
IRCBot	76%	-	-	-	13	-	3
IoTReaper	50%	-	-	-	-	-	2
Other (>14 families)	71%	13	120	5	6	1	45
Unclassified	70%	1	76	9	15	1	22
Total (weighted)	79%						

Table 6: Quantifying cross-talk: We find that 294 of the binaries communicate with their C&C server using communication protocols of other families as shown in the white cells and based on the Virustotal-based classification for DMain. The “Unclassified” category encompasses malware that Virustotal classifies as benign, which we are able to activate and impersonate their C&C server.

IP address	Single	2261
	Multiple	62
Domain	Fixed	257
	DGA	5

Table 7: The popularity of different techniques used to discover and connect to the C&C server for 2580 binaries. Note that some binaries use more than one technique.

we collect more than 2000 login-password combinations, which appear to correspond to default credentials for routers, IP cameras and DVRs e.g. root/hi3518 and root/x3511.

b. Profiling vulnerability exploits: 19 exploits. We find 166 binaries that attempt to exploit vulnerabilities on services running on different ports (e.g. HTTP or UPnP) by sending network packets once a session is created. We further investigated these attack vectors and identified 19 unique exploits. In some cases, we were able to associate these exploits to known vulnerabilities with Common Vulnerabilities and Exposures (CVE) identifiers, such as CVE-2017-8225 [51] and CVE-2018-10561 [52], with discovery dates spanning 7 years.

During the proliferation phase, we observe different models in selecting a potential IP address and port number for scanning. For example we observe in certain group of malware the scanning occurs on A.B.0.0/16 subnet while in some other case the IP address is randomly selected and does not limit to one specific subnet. Our goal in the future is to systematically explore the different models of IP and port number selection during the scanning phase of IoT malware.

Key Result 5: Observing cross-talk in 294 binaries. We identify 294 malware binaries that engage in “cross-talk” behavior: they use a C&C communication protocol from different IoT malware family than the one they belong to. Recall that our classification here is based on Virustotal, as discussed in Section 2.1. In Table 6, we show the number of activated binaries for six C&C communication protocols. Specifically, the columns correspond to a subset of widely used C&C communication protocols that are associated with four malware families. The rows of the table correspond to broad families of malware, which often consist of several variants

or sub-families. For example, Gafgyt consists of more than 6 sub-families, such as Lizekebab and Torlus. The coloring is supposed to make the relationships easier to see. The numbers in each colored cell indicate the number of malware binaries using the expected C&C communication protocol. For example, the numbers in soft blue (■) cells indicate Gafgyt binaries, which engage with one of the Gafgyt C&C server protocols.

The white cells in the table represent malware binaries that communicate with C&C protocols from other malware families. For example, we find 5 malware binaries that are classified as Gafgyt, but use the Mirai communication protocol (top right cell). We find 294 such binaries in total. We identify two possible reasons for this phenomenon: (a) there are “hybrid” malware that use communication protocols from other known malware families, or (b) the detection engines in Virustotal fail to correctly classify the malware. We found at least one hybrid binary among the 5 Gafgyt binaries mentioned above. This sample uses functions such as a telnet scanner, commonly used in the same malware family, while it uses the C&C communication protocol of the Mirai family.

5.3 Advanced malware behaviors

We highlight some of the interesting behaviors from our DMain malware dataset. Inspired by the MITRE ATT&CK framework [48], we show a break down of the common techniques used in each of the malware operation in Table 8 with the number of binaries exhibiting each behavior.

a. Evading detection via process masquerading: Process masquerading refers to the techniques that malware uses to change the name of malware-created processes to either: (a) the name of a common benign process, or (b) a randomly generated string. We observe 1494 binaries that employ this technique using the prctl system call.

b. Stopping remote access services: We observe that 413 malware binaries stop the HTTP, SSH, and Telnet services on the infected device. We suspect two possible reasons for this: (a) safeguarding against competing malware, and (b) avoiding detection or malware “clean-up” operations.

c. Cryptocurrency mining: We find 6 binaries that participate in a cryptocurrency mining botnet. Interestingly, none of these

Malware Procedure	Most common techniques					
	Bin.	Technique 1	Bin.	Technique 2	Bin.	Technique 3
Infection	1676	Brute-force login	166	Exploit public facing apps	-	None observed
Persistence	375	Add routine in rc script	333	Add a job to cronjob	15	Specific to IoT device
Defense evasion	1494	Process masquerading	648	Malware binary removal	128	Software packing
Identifying device	1445	Use network config	843	Use config files	286	List processes in device
Impact on host	414	Block OS level access	413	Stop remote services	6	Bitcoin mining

Table 8: The top three malware procedure and the related number of binaries that employ them in our malware binaries.

malware establish any connection to a potential C&C server during the 20 minute emulation interval. These binaries setup SOCKS proxy as a means of communicating with the rest of the botnet [14].

d. Unusual library file checks: Malware checks for the existence of library files in unusual directories and, based on their availability, the malware exhibits significantly different behaviors. For example, some binaries check for libnss in “/lib/mips-linux-gnu/”, which is usually not available in a native MIPS Linux machine. If available, it starts making a DNS requests to connect with its C&C server, otherwise it resorts to a fixed IP address.

e. Displacing prior malware: Some malware will check and attempt to supplant prior malware on the device. The malware searches for files and processes on the infected device as an indicator of a prior infection. In that case, the binary attempts to displace the other malware by killing the related processes and removing relevant files. For example, we find binaries that scan the list of processes for malware-related names such as daemon.mipsel.mod or lightaidra.

```
if "/bin/cfgmtd": # Ubiquiti device
    cfgmtd -w -p /etc/
    echo "wget http://${url}/nvr >
        /etc/persistent/rc.poststart"
    cfgmtd -w -p /etc/ && sleep 432000 && reboot
elif "/sbin/sncfg": # D-Link router
    /sbin/sncfg commit
    echo "wget ${url}/nvr"
    chmod +x /etc/custom.sh
    /sbin/sncfg commit
else: # basic device
    echo "* * * * /malware_name" > /etc/crontabs
```

Listing 1: Example of an “adaptive persistent process”; malware uses services specific to a device to make itself persistent by deploying a different technique per device.

f. Adaptive persistent process: We observe 7 malware binaries that use a sophisticated approach to make themselves persistent, which adapts to the type of device. Some of these binaries have customized behaviors for up to 8 types of device. In more detail, these binaries access a series of resources, which reveal the type of the device. For each device type, they have a different approach to make themselves persistent. In Listing 1, we show the pseudocode for 2 of the 8 configuration checks that a binary from the Tsunami family performs to determine the type of device and use the appropriate technique to make itself persistent. It checks for:

/bin/cfgmtd (used in Ubiquiti devices) or /sbin/sncfg (used in D-Link routers) to configure the NVRAM of the infected device. If none of these succeed, it will use the crontab process scheduler to make itself persistent by becoming a recurring process.

6 DISCUSSION AND FUTURE WORK

In this section, we discuss the usefulness and limitations of our approach.

a. Is our malware dataset representative? Finding a representative dataset is a hard problem in malware analysis research. In this work, we were able to amass a large archive of IoT malware binaries. We provide below intuitive arguments that give us a level of confidence in our dataset: (a) our archive includes binaries from all the reported IoT malware families [28], (b) Virustotal corroborates that our binaries are indeed IoT malware, and it spans at least 20 malware families, as we have mentioned earlier.

b. Will RIoTMAN work with new malware? Clearly, we cannot guarantee that RIoTMAN will work for all future IoT malware. However, we argue that it can efficiently create configurations within its capabilities, and it will continuously become more capable in analyzing all IoT malware as the more information is added to its Automation module. Furthermore, the 93% success rate with our DNew dataset is a promising indication.

c. Is an adaptive configuration creation necessary? We argue that an adaptive approach in creating analysis setup is more efficient and more effective. In traditional sandbox techniques, the analysis is done via pre-configured emulators [24], and as discussed in Section 4.1 this approach hinders with malware’s intended behavior. One could envision two different pre-configuration solutions. First, one could create an “all encompassing” emulator, but that would not work: many emulation choices will lead to file and resource conflicts. For example, we can not have an environment that encompass multiple standard libc libraries or multiple hardware peripherals of a certain type. Second, creating a plethora of pre-configured emulation instances is arguably tedious, less cost effective, and less efficient. As we showed in Section 4, a total of close to 700 unique emulation configurations were created to analyze all the malware binaries in an adaptive way combing configuration resources as needed. The millions of distinct IoT devices and their ever increasing number argue in favor of an adaptive approach.

d. Our platform as part of the anti-IoT-malware ecosystem. We see our platform as a building block in an ecosystem of tools and services that will attempt to understand, detect, and ultimately contain the effect of IoT malware and botnet. RIoTMAN can be engaged within a greater ecosystem: (a) **IoT Honeypots** collect

the binaries and execute them using RiOTMAN to execute and collect information about them; (b) **Public and private sources**, such as security researchers and online resources can analyze collected binaries using our platform for more information and validation; (c) **Domain reputation services**, such as Google's blacklist and Web of Trust, can use our platform to identify domains and DGA based domains that are being utilized by IoT malware; (d) **IP address blacklists** can exchange information with our platform regarding the reputation of IP addresses; (e) **Firewall and NIDS** can derive rules and signatures regarding suspicious network behavior by using the dynamic behavior of an IoT malware. (f) **Traffic monitoring repositories and services** can use RiOTMAN to identify the potential volumetric network traffic generated by botnet of infected IoT malware.

7 RELATED WORK

We review related work in the following broad categories.

a. Emulation approaches for IoT malware: There are several open source sandboxes, such as Limon [47], Detux [27], and cuckoo [34], which provide a dynamic analysis capability for Linux malware. However, they lack support for IoT specific malware, e.g. ARM and MIPS architectures. In addition, they do not support engagement with sandbox instances at the networking layer.

Recently, Padawan [24], an online malware analysis service, was introduced for analyzing Linux malware binaries. It profiles the malware behavior at the OS level, but it does not focus on IoT malware or networking behavior. The approach is philosophically different from ours, as it creates a large number of pre-configured instances of devices. In addition, a recent effort, RARE [26], focuses on routers, a subset of IoT devices, and does not provide any automated engagement capability as we do in this work.

To the best of our knowledge, none of the above approaches combines the following elements: (a) building the IoT device configuration iteratively, and (b) providing automated capability to communicate with the malware

b. Measurement studies of malware traffic in the wild: Several studies analyze the large scale effects of real attacks using observed data [29, 42, 44–46]. A recent study studies the Mirai botnet, which was enabled by IoT malware [10]. These efforts differ in nature from our work, which is binary-centric: we assume no access to measured network data.

c. Non-malware IoT network traffic analysis: Several recent works measure and analyze IoT devices to identify patterns malicious behavior [6, 13, 19, 50]. In addition, other efforts focus on the privacy concerns introduced by such devices [3, 56]. By contrast, we focus on dynamic analysis of IoT malware in this work.

d. Embedded device firmware analysis: Several efforts focus on identifying vulnerabilities in the firmware of embedded devices using static [23, 31, 70] and dynamic analysis [18]. These efforts are complementary to our work, as we do not focus on identifying vulnerabilities on the device.

e. Dynamic analysis of non-IoT malware: There is a extensive literature of dynamic analysis studies for malware, which focus on PC and Android malware [39–41, 43]. As we discussed in the introduction, IoT malware poses several challenges and requires novel approaches and tools.

f. Static analysis of malware: The majority of static analysis efforts target PC-based and smartphone malware. In addition, using only static analysis for malware binaries has known limitations [49]. Several efforts develop techniques to understand the structure of the binary [7, 20]. Other efforts use symbolic execution to explore the execution path of binaries, such as project angr [61].

8 CONCLUSION

Our approach is conceptually a novel way to enable the dynamic analysis of IoT malware at scale and with minimal manual effort. It tackles head-on the key challenge: the difficulty in identifying the target device configuration.

Our main contribution in the combined effect of two key novelties: (a) Iterative Adaptation, and (b) Automated Engagement. First, our platform employs an iterative process that incrementally “builds” the configuration of malware’s target device. Second, our platform minimizes the need for manual effort: it automates our interaction with the malware during the Iterative Adaptation and the C&C server impersonation phase. Our platform also combines in-depth analysis at the system call and the network level, which helps with understanding the intention behind the malware activities. Our platform is successful at both activating and engaging with the malware binaries. We activate 93% of our binaries and manage to impersonate the C&C server for 79% of our binaries.

By open-sourcing our approach, we expect to establish a community-driven capability in the fight against IoT malware. We view RiOTMAN as an important step towards an effective dynamic analysis capability for IoT malware.

ACKNOWLEDGEMENT

We would like to thank our anonymous reviewers for their insightful feedback. We thank our anonymous shepherd for their constructive comments that helped shape up this paper. This work was supported by the UC Multicampus National Lab Collaborative Research and Training (UCNLCRT) award #LFR18548554.

REFERENCES

- [1] Permanent denial-of-service attack: Brickerbot. <https://ics-cert.us-cert.gov/alerts/ICS-ALERT-17-102-01A> (2017-04).
- [2] Openwrt linux distribution for embedded devices | soho routers. <https://openwrt.org/> (2019-02).
- [3] Iot inspector. <https://iotinspector.org/> (2020-06).
- [4] Padawan. <https://padawan.s3.eurecom.fr> (2020-06).
- [5] ALEX.TURING, G. Y. An Analysis of Godlua Backdoor, 2019. <https://blog.netlab.360.com/an-analysis-of-godlua-backdoor-en/>.
- [6] ALRAWI, O., LEVER, C., ANTONAKAKIS, M., AND MONROSE, F. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE Symposium on Security and Privacy (SP)* (2019), IEEE, pp. 1362–1380.
- [7] ANDERSON, B., QUIST, D., NEIL, J., STORLIE, C., SEAMAN, C., SULLIVAN, N., THOMAS, K., AND ZHOU, Y. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)* (2017), USENIX Association.
- [8] ANGRISHI, K. Turning internet of things (iot) into internet of vulnerabilities (ioV): Iot botnets. *arXiv preprint arXiv:1702.03681* (2017).
- [9] ANNA-SENPAI. [free] world’s largest net:mirai botnet, client, echo loader, cnc source code release. <https://hackforums.net/showthread.php?tid=5420472>. (2016).
- [10] ANTONAKAKIS, M., APRIL, T., BAILEY, M., BERNHARD, M., BURSSTEIN, E., COCHRAN, J., DURUMERIC, Z., HALDERMAN, J. A., INVERNIZZI, L., KALLITSIS, M., KUMAR, D., LEVER, C., MA, Z., MASON, J., MENSCHER, D., SEAMAN, C., SULLIVAN, N., THOMAS, K., AND ZHOU, Y. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)* (2017), USENIX Association.
- [11] BADER, J. Domain generation algorithm. <https://www.johannesbader.ch/2015/05/the-dga-of-ranbyus/> (2015).
- [12] BELLARD, F. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track* (2005), pp. 41–46.

- [13] BHATIA, R., BENNO, S., ESTEBAN, J., LAKSHMAN, T., AND GROGAN, J. Unsupervised machine learning for network-centric anomaly detection in iot. In *Proceedings of the 3rd ACM CoNEXT Workshop on Big Data, Machine Learning and Artificial Intelligence for Data Communication Networks* (2019), pp. 42–48.
- [14] BIJMAN, H. L., BOOIJ, T. M., AND DOERR, C. Just the tip of the iceberg: Internet-scale exploitation of routers for cryptojacking. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019), pp. 449–464.
- [15] CABALLERO, J., POOSANKAM, P., KREIBICH, C., AND SONG, D. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM conference on Computer and communications security* (2009), ACM, pp. 621–634.
- [16] CABALLERO, J., YIN, H., LIANG, Z., AND SONG, D. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In *Proceedings of the 14th ACM conference on Computer and communications security* (2007), ACM, pp. 317–329.
- [17] CETIN, F., HERNANDEZ GANAN, C., ALTENA, E., KASAMA, T., INOUE, D., TAMIYA, K., TIE, Y., YOSHIOKA, K., AND VAN EETEN, M. Cleaning up the internet of evil things: Real-world evidence on isp and consumer efforts to remove mirai.
- [18] CHEN, D. D., WOO, M., BRUMLEY, D., AND EGELE, M. Towards automated dynamic analysis for linux-based embedded firmware. In *NDSS* (2016).
- [19] CHEN, J., DIAO, W., ZHAO, Q., ZUO, C., LIN, Z., WANG, X., LAU, W. C., SUN, M., YANG, R., AND ZHANG, K. Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing. In *NDSS* (2018).
- [20] CHRISTODORESCU, M., AND JHA, S. Static analysis of executables to detect malicious patterns. Tech. rep., Wisconsin Univ-Madison Dept of Computer Sciences, 2006.
- [21] CHRISTODORESCU, M., JHA, S., AND KRUEGEL, C. Mining specifications of malicious behavior. In *Proceedings of the 1st India software engineering conference* (2008), ACM, pp. 5–14.
- [22] COSTIN, A., ZADDACH, J., FRANCILLON, A., BALZAROTTI, D., AND ANTIPOLIS, S. A large-scale analysis of the security of embedded firmwares. In *USENIX Security Symposium* (2014), pp. 95–110.
- [23] COSTIN, A., ZARRAS, A., AND FRANCILLON, A. Automated dynamic firmware analysis at scale: a case study on embedded web interfaces. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security* (2016), ACM, pp. 437–448.
- [24] COZZI, E., GRAZIANO, M., FRATANTONIO, Y., AND BALZAROTTI, D. Understanding linux malware. In *IEEE Symposium on Security & Privacy* (2018).
- [25] CUI, W., PEINADO, M., CHEN, K., WANG, H. J., AND IRUN-BRIZ, L. Tupni: Automatic reverse engineering of input formats. In *Proceedings of the 15th ACM conference on Computer and communications security* (2008), ACM, pp. 391–402.
- [26] DARKI, A., CHUANG, C.-Y., FALOUTSOS, M., QIAN, Z., AND YIN, H. Rare: A systematic augmented router emulation for malware analysis. In *International Conference on Passive and Active Network Measurement* (2018), Springer, pp. 60–72.
- [27] DETUX. Github. The Multiplatform Linux Sandbox, May 13 2015.
- [28] F-SECURE. Iot threat landscape: old hacks, new devices. <https://bit.ly/2vMIFSL> (2019).
- [29] FARINHOLT, B., REZAEIRAD, M., PEARCE, P., DHARMASANI, H., YIN, H., LE BLOND, S., MCCOY, D., AND LEVCHENKO, K. To catch a ratter: Monitoring the behavior of amateur darkcomet rat operators in the wild. In *2017 38th IEEE Symposium on Security and Privacy (SP)* (2017), IEEE, pp. 770–787.
- [30] FBI. Cyber actors use internet of things devices as proxies for anonymity and pursuit of malicious cyber activities. <https://www.ic3.gov/media/2018/180802.aspx> (2018).
- [31] FENG, Q., ZHOU, R., XU, C., CHENG, Y., TESTA, B., AND YIN, H. Scalable graph-based bug search for firmware images. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 480–491.
- [32] GNU. Gnu binutils. <https://www.gnu.org/software/binutils/> (2019).
- [33] GOASDUFF, L. Gartner says 5.8 billion enterprise and automotive iot endpoints will be in use in 2020. Online: <https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-iot> (2019).
- [34] GUARNIERI, C., TANASI, A., BREMER, J., AND SCHLOESSER, M. The cuckoo sandbox, 2012.
- [35] GUSTAFSON, E., MUENCH, M., SPENSKY, C., REDINI, N., MACHIRY, A., FRATANTONIO, Y., BALZAROTTI, D., FRANCILLON, A., CHOE, Y. R., KRUEGEL, C., ET AL. Toward the analysis of embedded firmware through automated re-hosting. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID)* (2019) (2019), pp. 135–150.
- [36] HERWIG, S., HARVEY, K., HUGHEY, G., ROBERTS, R., AND LEVIN, D. Measurement and analysis of hajime, a peer-to-peer iot botnet. In *NDSS* (2019).
- [37] HEX-RAYS, S. Idapro disassembler, 2008.
- [38] INTERPOL. 'internet of things' cyber risks tackled during interpol digital security challenge. <https://bit.ly/2LleIwR> (2018).
- [39] KIRAT, D., AND VIGNA, G. Malgene: Automatic extraction of malware analysis evasion signature. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 769–780.
- [40] KOLBITSCH, C., COMPARETTI, P. M., KRUEGEL, C., KIRDA, E., ZHOU, X.-Y., AND WANG, X. Effective and efficient malware detection at the end host. In *USENIX security symposium* (2009), pp. 351–366.
- [41] KOLBITSCH, C., HOLZ, T., KRUEGEL, C., AND KIRDA, E. Inspector gadget: Automated extraction of proprietary gadgets from malware binaries. In *Security and Privacy (SP), 2010 IEEE Symposium on* (2010), IEEE, pp. 29–44.
- [42] KWON, B. J., MONDAL, J., JANG, J., BILGE, L., AND DUMITRAS, T. The dropper effect: Insights into malware distribution with downloader graph analytics. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 1118–1129.
- [43] LANZI, A., BALZAROTTI, D., KRUEGEL, C., CHRISTODORESCU, M., AND KIRDA, E. Accessminer: using system-centric models for malware protection. In *Proceedings of the 17th ACM conference on Computer and communications security* (2010), ACM, pp. 399–412.
- [44] LEITA, C., AND DACIER, M. Sgnet: a worldwide deployable framework to support the analysis of malware threat models. In *2008 Seventh European Dependable Computing Conference* (2008), IEEE, pp. 99–109.
- [45] LEITA, C., MERMOUD, K., AND DACIER, M. Scriptgen: an automated script generation tool for honeyd. In *21st Annual Computer Security Applications Conference (ACSAC'05)* (2005), IEEE, pp. 12–pp.
- [46] LEVER, C., KOTZIAS, P., BALZAROTTI, D., CABALLERO, J., AND ANTONAKAKIS, M. A lustrum of malware network communication: Evolution and insights. In *Security and Privacy (SP), 2017 IEEE Symposium on* (2017), IEEE, pp. 788–804.
- [47] LIMON. Github. Sandbox for Analyzing Linux Malwares, May 13 2015.
- [48] MITRE. Mitre att&ck. <https://attack.mitre.org/> (2019).
- [49] MOSER, A., KRUEGEL, C., AND KIRDA, E. Limits of static analysis for malware detection. In *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual* (2007), IEEE, pp. 421–430.
- [50] NGUYEN, D. T., SONG, C., QIAN, Z., KRISHNAMURTHY, S. V., COLBERT, E. J., AND MCDANIEL, P. Iotsan: Fortifying the safety of iot systems. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies* (2018), pp. 191–203.
- [51] NIST. CVE-2017-8225. Available from MITRE., 25 2017.
- [52] NIST. CVE-2018-10561. Available from MITRE., May 3 2018.
- [53] OOSTERHOF, M. Cowrie ssh/telnet honeypot. <https://github.com/cowrie/cowrie> (2014).
- [54] PENG, P., YANG, L., SONG, L., AND WANG, G. Opening the blackbox of virus-total: Analyzing online phishing scan engines. In *Proceedings of the Internet Measurement Conference* (2019), pp. 478–485.
- [55] RADARE. Reverse engineering framework and commandline tools. <https://github.com/radare/radare2> (2019).
- [56] REN, J., DUBOIS, D. J., CHOFFNES, D., MANDALARI, A. M., KOLCUN, R., AND HADDAI, H. Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach. In *Proceedings of the Internet Measurement Conference* (2019), pp. 267–279.
- [57] RODRIGUES, B. Luabot: Malware targeting cable modems, 2016.
- [58] ROKON, M. O. F., ISLAM, R., DARKI, A., PAPALEXAKIS, E. E., AND FALOUTSOS, M. Sourcefinder: Finding malware source-code from publicly available repositories in github. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)* (San Sebastian, Oct. 2020), USENIX Association, pp. 149–163.
- [59] ROSSOW, C., DIETRICH, C. J., BOS, H., CAVALLARO, L., VAN STEEN, M., FREILING, F. C., AND POHLMANN, N. Sandnet: Network traffic analysis of malicious software. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security* (2011), ACM, pp. 78–88.
- [60] SHAILA, G. S., DARKI, A., FALOUTSOS, M., ABU-GHAZALEH, N., SRIDHARAN, M., ET AL. Idapro for iot malware analysis? In *12th {USENIX} Workshop on Cyber Security Experimentation and Test ({CSET})* 19 (2019).
- [61] SHOSHITAISHVILI, Y., WANG, R., SALLS, C., STEPHENS, N., POLINO, M., DUTCHER, A., GROSEN, J., FENG, S., HAUSER, C., KRUEGEL, C., AND VIGNA, G. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *IEEE Symposium on Security and Privacy* (2016).
- [62] STONE-GROSS, B., COVA, M., CAVALLARO, L., GILBERT, B., SZYDLOWSKI, M., KEMMERER, R., KRUEGEL, C., AND VIGNA, G. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security* (2009), ACM, pp. 635–647.
- [63] TALOS, C. New vpnfilter malware targets at least 500 k networking devices worldwide, 2019.
- [64] THEWHITETEAM. Linux.wifatch. https://gitlab.com/rav7teif/linux.wifatch/project_members (2019).
- [65] TOTAL, V. Virustotal-free online virus, malware and url scanner. Online: <https://www.virustotal.com/en> (2012).
- [66] ULLRICH, J. B. Linksys worm "themoon" summary: What we know so far. goo.gl/giHF7Q (2014).
- [67] VERVIER, P.-A., AND SHEN, Y. Before toasters rise up: A view into the emerging iot threat landscape. In *International Symposium on Research in Attacks, Intrusions, and Defenses* (2018), Springer, pp. 556–576.
- [68] WALDNER. Tun/tap interface tutorial. <https://backreference.org/2010/03/26/tuntap-interface-tutorial/> (2010).
- [69] WATSON, D., AND RIDEN, J. The honeynet project: Data collection tools, infrastructure, archives and analysis. In *2008 WOMBAT Workshop on Information Security*

- Threats Data Collection and Sharing* (2008), IEEE, pp. 24–30.
- [70] XU, X., LIU, C., FENG, Q., YIN, H., SONG, L., AND SONG, D. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 363–376.
 - [71] ZADDACH, J., BRUNO, L., FRANCILLON, A., AND BALZAROTTI, D. Avatar: A framework to support dynamic security analysis of embedded systems' firmwares. In *NDSS* (2014).
 - [72] ZHENG, Y., DAVANIAN, A., YIN, H., SONG, C., ZHU, H., AND SUN, L. Firm-afl: high-throughput greybox fuzzing of iot firmware via augmented process emulation. In *28th {USENIX} Security Symposium ({USENIX} Security 19)* (2019), pp. 1099–1114.