UNIVERSITY OF CALIFORNIA
RIVERSIDE

Towards a Systematic Analysis of IoT Malware

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Ahmad Darki

September 2020

Acknowledgments

The work presented in this dissertation would not have been possible without the help of a big number of people who have supported me and my decisions in pursuing PhD. I would like to start with expressing my sincerest gratitude and appreciation to my PhD advisor Prof. Michalis Faloutsos. Prof. Faloutsos's advice and support taught me to think critically not only about my PhD work but also about my everyday life. I know I will always be grateful for this opportunity and remember the lessons I learnt for the rest of my life. It is unfortunate that I was not able to have an opportunity to say these words in person, due to COVID19. *Thank you Michalis, and I hope to see you again soon.*

I am thankful for the committee members, Prof. Nael Abu-Ghazaleh Prof. Srikanth Krishnamurthy, and Prof. Heng Yin for their constructive comments and inputs. Their comments helped shape up my dissertation and the research topic. Also, I would like deeply thank Prof. Zhiyun Qian for his insightful comments in the beginning of my dissertation.

I would like to thank my coauthors who have helped me in publications: Prof. Vagelis Papalexakis, Prof. Manu Sridharan, Prof. Spiros Mancoridis, Sri Shaila G, Md Omar Faruk Rokon, Risul Islam, and Alexander Duff.

PhD is a long journey with countless hurdles which deeply affects the mental health and well being of students including me. There is a great number of people that have supported me with this journey and my decisions. If I could have the space to name all of them, that would lead to an additional 30 pages. However I will try to name very few anyways. I would like to start with my good friends: Ali Zoghi, Joobin Gharibshah, Alireza Shahsavari, Javad Darivandpou, Amir Ardalan Rezaei, Ashkan Yousefpour, Masoud

Vafabakhsh, Hodjat Asghari Esfeden, Shahriar Afzal, Aref Asvadi, Ali Davanian, Adel Karimi, Pravallika Devineni, Kittipat Apicharttrisorn, Ryan and Tabby Stanton, Shailendra Singh, Saheli Ghosh, Yue Cao, Chun-Yu Chuang, Jason Ott, Ravdeep Pasricha, Azeem Aqil, Ahmed Atya, Dao Anh Tuan, Aditya Dhakal, Shitong Zhu, Pengxiong Zhu, Yizhuo Zhai, Shasha Li, Zhongjie Wang, and many many more. I am thankful that I have met you and sad that I did not get to celebrate my graduation with you. I am hopeful that our paths will cross again.

Finally, I would like to thank the people who are closer to my heart and I care the most. I would like to start with my girlfriend, María, who too is a PhD student. Regardless of pressure of being a PhD, she never ceased to support me, care for me, and love me. *I love you and thank you for everything.* Lastly, my family. I left them in 2014 and have not seen them ever since due to U.S. visa restrictions imposed on Iranian citizens. Although so far away and so long ago, but they never stopped caring for me. The immense amount of love and support was what kept me going for the past years and more years to come. I love them and I hope to see them again.

To my love Maria and my family for their endless support.

Mom, Dad, Sabrin, Ala, and Mohammad Amin I love you and I miss you all so much.

ABSTRACT OF THE DISSERTATION

Towards a Systematic Analysis of IoT Malware

by

Ahmad Darki

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2020
Prof. Michalis Faloutsos, Chairperson

Internet of Things (IoT) malware established itself as the new type of threat after enabling the most intense DDoS attacks to date using Mirai botnet. All indications suggest that the problem will become more acute. First, there is widely-available source code of IoT malware such as Mirai and BASHLITE making it easy for BlackHat hackers to create their own botnet. Second, such malware employ capabilities to target particular group of IoT devices and perform different malicious operations such as harvesting network traffic in routers. Third, there is evidence of IoT malware getting better: new families appear and existing families evolve and adopt sophisticated techniques, including proliferation techniques, and types of C&C discovery mechanisms.

In the first Chapter, we tackle the problem of malware attacking home routers. Router-specific malware has emerged as a new vector for hackers, but has received relatively little attention compared to malware on other devices. We propose, RARE, a systematic approach to analyze router malware and profile its behavior focusing on home-office routers. The key novelty is the intelligent augmented operation of our emulation that manages to fool malware binaries to activate irrespective of their target platform. RARE has the ability to: (a) instantiate an emulated router with or without malware, (b) replay arbitrary network traffic, (c) monitor and interact with the malware in a semi-automated way.

In the second Chapter, we develop RIoTMAN, a comprehensive emulation and dynamic analysis platform for IoT malware. RIoTMAN can activate the malware and communicate with it to explore its spectrum of behaviors. The power of our platform lies on two key novelties: (a) Iterative Adaptation, and (b) Automated Interaction.

In the third Chapter, we perform longitudinal study on all IoT malware to analyze their behavior on the host and networking level. In this study, we break down the malware techniques and tactics inspired by MITRE ATT&CK framework. We profile the techniques that the IoT malware employs to communicate with the botnet, recruiting devices, and the protocol used to communicate with the C&C server. Moreover, by impersonating their server, we issue control commands for the malware to enter its proliferation phase or start a DoS attack. One of the outcome of our study is the TBs of attack traffic from real IoT malware which can be further used in related studies.

Lastly, we develop techniques to explore IoT malware behavior under different analysis environment configuration. First, we identify the techniques that a given IoT malware uses to identify the target environment. Second we perform dynamic executions under different target platform using RIoTMAN and profile changes in the malware behavior. We identify malware that exhibit 8 distinct behavior depending on the target environment.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With their quick emergence, Internet of Things (IoT) devices have established themselves as a unique addition to the cyberspace. Their lightweight design have made them a favorable approach in producing sensors and single purpose devices in mass. Gartner anticipates the number of Internet-connect IoT devices to exceed 5.8 billion by the end of the year 2020 [81]. This preference by device manufacturers introduces opportunities for fast growth. However, this also raises concerns about the security practices that have been taken into consideration during their design.

In an era where the Internet is inundated by IoT devices, the security implications brings about an opportunity for criminals to take advantage of the abundance of such device. The lack of secure software development practices has enabled numbers of software vulnerabilities embedded in their IoT device firmware [56].

The number of attacks on IoT devices has exponentially grown in the past few years [72]. We observed the largest DDoS attack recorded being carried out by Mirai botnet of infected IoT devices [20]. In addition, with the release of botnet malware source code [129] it has made it possible for black hat hackers to deploy more advanced botnet. For example, TheMoon malware in February 2014 consisted of simple C&C infrastructure and a few proliferation techniques which later in May 2018 was redesigned to exploit newer vulnerabilities [143] and had a more sophisticated botnet architecture [191]. All indication suggest that this threat continues to become more acute.

Problem Definition

The overarching problem we address in this dissertation is: "How can we execute and analyze IoT malware effectively and efficiently?" More specifically, given an IoT malware we want to extract information about its target environment and its OS and network behavior. We propose and implemented tools and techniques to achieve our goal of executing and analyzing their behavior. We propose novel approaches to automatically identify the target platform of a given IoT malware and analyze its dynamic behavior. We conduct longitudinal analysis to automatically engage with the IoT malware as its C&C server to explore its malicious functions.

Challenges and Assumptions

Following we list the main challenges faced in analyzing IoT malware:

Activating the malware: The plethora of diverse set of IoT devices makes the problem of analyzing IoT malware more challenging. Compared with PC and smart phone malware, IoT malware targets a wide range of devices running on different CPU architectures (such as MIPS, ARM, and PowerPC) under different versions of Linux kernel.

Almost all IoT devices have vendor specific hardware peripherals that can not be simulated. Activating a given IoT malware requires knowledge about its target platform and simulating that is not a trivial effort.

Reducing manual effort: In addition to correctly identifying the target platform for a given IoT malware another challenge rises with large scale analysis. Given their rapid growth, we are faced with a large number of IoT malware that require automated malware analysis. Automatically identifying and creating the target platform for a large number of IoT malware is a challenging problem.

Engage in communication: For an in-depth analysis, the goal is to make all the resources available while analyzing a given malware. This includes enabling network communication and engaging in communication with the malware. The challenge is to simulate a realistic networking environment for the malware to both contain its spread and engage in communication with it.

Sensitive IoT malware: More complex IoT malware threat has been observed to enable state sponsored Advanced Persistent Threat (APT) attacks. Such malware and similar ones are crafted to start activating or engage in specific activities if the target environment hosts certain criteria. Identifying such criteria and observing change in the malware behavior is not a trivial task and requires expensive and heavy in-depth analysis.

Related Work

Although a significant amount of work has been done in analyzing malware behavior, however, there is little research in the space of IoT malware. We group the overarching related work in the following categories:

(i) Linux sandboxes and emulators [120, 65, 161]; (ii) studies of malware traffic in the wild [43, 20, 119]; (iii) IoT binary and firmware analysis [46, 56, 204, 88]; (iv) studies on malware behavior analysis [115, 109, 53, 52, 158, 159, 28]; (v) studies on environment sensitive malware [27, 104, 121, 116, 106, 107, 77, 147, 199, 132, 152, 110, 198]. In each chapter we discuss the related work in more depth.

In this dissertation, we propose novel techniques to analyze the malware targeting IoT devices. We developed tools and analysis environment to perform dynamic and static analysis on the malware binaries and record their execution traces. We profile malware infection process, proliferation techniques, C&C server communication protocols, and other malicious activity.

The flow of this dissertation is as follows. In Chapter 2, we tackle the problem of malware attacking home routers by creating tools and proposing techniques to identify an infected router. In Chapter 3, we develop an IoT malware analysis tool to activate and engage with any given malware in an automatic way and record its execution traces. In Chapter 4, we perform a longitudinal analysis on more than 3000 IoT malware to to profile their behavior in both OS and networking level. Finally, in Chapter 5, we propose techniques to analyze IoT malware sensitive to target environment and identify new behaviors that are not observable otherwise. Following, we highlight the motivation and the novelty of each Chapter.

RARE: Riverside's Augmented Router Emulation

Our goal in Chapter 2 is to analyze and profile the behavior of a router malware. Router-specific malware has emerged as a new vector for hackers, but has received relatively little attention compared to malware on other devices. A key challenge in analyzing router malware is getting it to activate, which is hampered by the diversity of firmware of various vendors and a plethora of different platforms. We propose, RARE, a systematic approach to analyze router malware and profile its behavior focusing on home-office routers. The key novelty is the intelligent augmented operation of our emulation that manages to fool malware binaries to activate irrespective of their target platform. This is achieved by leveraging two key capabilities: (a) a static level analysis that informs the dynamic execution, and (b) an iterative feedback loop across a series of dynamic executions, whose output informs the subsequent executions. From a practical point of view, RARE has the ability to: (a) instantiate an emulated router with or without malware, (b) replay arbitrary network traffic, (c) monitor and interact with the malware in a semi-automated way. We evaluate our approach using 221 router-specific malware binaries. First, we show that our method works: we get 94% of the binaries to activate, including obfuscated ones, which is a nine-fold increase compared to the 10% success ratio of the baseline method. Second, we show that our method can extract useful information towards understanding and profiling the botnet behavior: (a) we identify 203 unique IP addresses of C&C servers, and (b) we observe an initial spike and an overall 50% increase in the number of system calls on infected routers.

RIoTMAN: Riverside's IoT Malware Analysis

The motivating question in Chapter 3 is how can we conduct dynamic analysis on a given IoT malware efficiently? A key challenge is that such malware target a plethora of different devices, which makes finding the target device not trivial. This problem does not appear nearly as much in PC and smartphones malware, where devices are more uniform. The contribution of our work is two fold: (a) we develop RIoTMAN, a comprehensive emulation and dynamic analysis method, and (b) we study the dynamic behavior of 2885 IoT malware systematically. The power of our platform lies on two key novelties: (a) Iterative Adaptation, and (b) Automated Engagement. First, RIoTMAN employs an intelligent iterative process that incrementally "builds" the configuration of the target device. Second, our platform automates the interaction with the malware during the Iterative Adaptation and C&C server impersonation phase. We have studied more than 2885 malware binaries that encompass major IoT malware identified between July 2014 and January 2018. First, we achieve an activation rate of 93% of the binaries which includes 173 malicious samples, which are reported as benign by Virustotal. Second, we engage with malware 79% of the binaries impersonating their C&C server: we make the malware initiate a DDoS attack, or enter its proliferation mode. Finally, we study and document some ingenious techniques that malware uses, including detecting the emulation environment, safeguarding against anti-virus methods, and making itself persistent.

A Longitudinal Study of IoT Malware Behavior

Understanding the techniques that malware uses to exploit the resources on an IoT device and communicate with its server (or the rest of the botnet), helps with identifying the best practice to contain such threat and further mitigation. Most effort in malware analysis has focused on the behavior of the malware on PC and smartphone platform and little work has focused on IoT malware

In Chapter 4, we conduct an extensive study on IoT malware binaries collected from the wild to understand: 1) the techniques and tactics they use to exploit the infected IoT device, 2) the proliferation techniques that IoT malware uses, 3) different communication protocols that are used to operate the malware. We perform exhaustive automated dynamic analysis on 2885 malware binaries to study their behavior. We show: a) a breakdown of the IoT malware techniques used for exploiting an infected device which speaks volumes about the differences between IoT malware and that of a PC or smartphone, b) 2 host scanning techniques that can be attributed to botnets characteristics i.e. either a botnet with more infected devices or one infecting more diverse sets of devices, and c) by identifying and impersonating their server, we are able to issue control commands to trigger different denial of service (DoS) functions as well as instructing the malware to reconfigure or terminate itself.

We have collected magnitudes of TB of real attack traffic from our simulated environment and which we make available for the community.

Systematic Analysis of IoT Malware Behavior in Target Environment

One of the critical challenges in analyzing malware is to observe its behavior in an emulated target environment. However, emulating all possible environments is not a feasible task given the plethora of IoT devices.

In Chapter5, we present a novel and systematic approach to analyze IoT malware with polymorphic behavior under different targeted environment. First, we identify the techniques that malware uses to inquire the type of target environment. Second, we perform dynamic execution under the different target environment and profile the changes in the malware behavior. We show case the capability of our approach using real world IoT malware and identify 3 main type of malware: 1) malware that employs "query and infect" technique to verify the target environment and infect it: for example a set of 11 malware binaries exhibit 8 different infection process based on the target environment; 2) malware that have "split" personality: malware identifies artifacts of analysis environment and stop its execution 3) malware with "hidden" behavior: malware that reveals additional behavior upon identifying the specific targeted environment.

Our findings in this work is an indication that IoT malware analysis requires more scrutiny with respect to its targets.

Chapter 2

Systematic Augmented Router Emulation for Malware Analysis

Compromising routers is emerging as a new type of threat with potentially devastating effects [148]. For example, on October 2016, in Mirai botnet attack there has been a series of DDoS attacks on *Dyn, Inc.* servers using IoT devices including routers [20]. The lack of mature protection technologies makes this a fertile ground for attacks. We argue that a compromised router provides significant new capabilities to an attacker, beyond those of a compromised end-device. By compromising a router, the attacker can: (a) access or block the network packets going through it, (b) steal cookies and session IDs[149] to impersonate the user or compromise her privacy, and (c) hijack and redirect communication via a DNS redirection to rogue DNS server.

Attacking and protecting routers is significantly different compared to laptops and desktops, therefore new methods and tools are needed. First, routers have limited resources

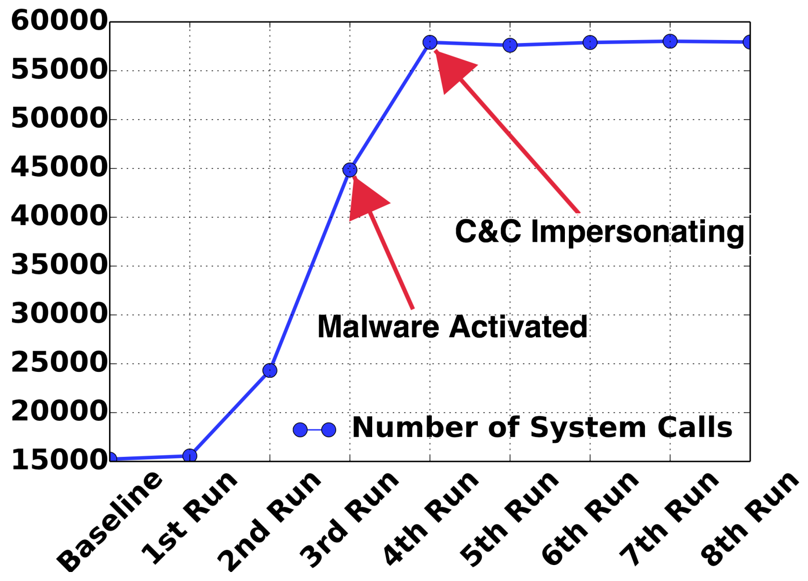


Figure 2.1: RARE gets the router malware to activate and communicate with the C&C server in the 3^{rd} run of its iterative operation. We plot the number of system calls for each 1300 seconds of a run. The baseline approach is shown first. The 1^{st} run of RARE is an emulation informed by the static analysis only. Each subsequent run is informed by the previous run.

in terms of CPU, and memory. Therefore, malware developers have less resources in their disposition. However, the same challenges apply to the security solutions as well. Second, routers have variable device configurations [90] that decreases the applicability of both malware, and system analysis tools. The former is a challenge for malware authors, while the latter is a challenge for any emulation capability, which needs to pretend to be many different configurations in order to get firmware and malware to run. Finally, many, if not most, router firmware are proprietary, and thus difficult to emulate [46].

Our goal in this work is to fully understand router malware binaries and their operation focusing on off-the-self home-office routers.The desired output of this work is two-fold. First, we want to analyze the malware in order to create an environment that will “fool” it to activate and reveal its behavior. Second, we want to profile and distinguish the behavior of an infected router from that of a benign one. The overarching challenge is the plethora of proprietary firmware and hardware router configurations, as we mentioned above. In addition, there is a scarcity of tools for static analysis for MIPS and ARM architectures, which are the most common platforms for routers. Such tools could have helped inform the emulation environment. As we will see below, a straightforward emulation attempt could have a very low success rate in fooling the malware to activate.

Router malware has received relatively little attention compared to malware on other devices. We can distinguish the following areas of research: (a) developing emulator capabilities [30, 92, 71, 178, 71, 178], (b) vulnerability analysis for embedded devices, [56, 46, 55, 76], and (c) malware analysis focusing on PC and smartphone based malware [52, 115, 109, 110]. We discuss related work in section 2.4.

We propose, RARE (Riverside’s Augmented Router Emulator), a systematic approach to analyze router malware and understand its behavior in depth. The key novelty is the augmented operation of our approach, which fools malware binaries to activate irrespective of their preferred target platform. In other words, instead of trying to guess the right router platform for each malware, we start with a generic one and we carefully and iteratively “adapt” it to fool the malware. This is achieved by leveraging two key capabilities: (a) a static level analysis that informs the dynamic execution, and (b) an iterative feedback loop across a series of dynamic runs, the output of which informs the subsequent run. We provide an overview of our approach in Figure 2.2.

From a practical point of view, RARE has the ability to run the malware on an emulated router and consists of the following modules, whose goal is to: a) extract information for malware execution by analyzing the binary statically, b) create an emulated router enhanced with the appropriate configurations that a malware needs for its execution, c) inject malware into the router and fool it to activate, d) replay pre-recorded network traffic with crafted C&C responses to malware requests, e) enhance the emulation using information derived from the previous runs. This process works in an iterative fashion to enhance the emulation using information from previous runs in order to have the malware to activate itself.

We evaluate our approach using 221 router-specific malware binaries from a community-based project, which requested to be anonymous. Our results can be summarized in the following points.

a. Achieving 94% malware activation success ratio, and 88.8% for obfuscated malware. We show that our system is successful in fooling malware to activate, as **94%** of our binaries become active. We say that a malware binary activates, when the malware attempts to communicate with the C&C server. By contrast, an emulation without any of our augmented functions, which we refer to as **baseline**, can activate only 10% of the binaries. Furthermore, our approach manages to activate 88.8% of our obfuscated binaries, which are the types of binaries that are especially crafted to outplay static analysis techniques. We show the iterative operation of our approach in Figure 2.1, where we plot the number of system calls for each run. Note that the malware is activated within a sandboxed environment and thus does not pose any threat to real devices

b. Extracting useful information: IP addresses, and domains. Having this powerful capability, we are able to extract useful information for the malware. We find **203 malicious IP addresses and domains**, and we naturally consider these addresses to be malicious, since they are or have been used for botnet communications. Note that using just static analysis, we find less than 25% of these C&C communication addresses and domain names.

c. Developing infected router profiles. We identify features for detecting infected routers. Specifically, we observe an initial spike and a 50% increase in the number of system calls in infected routers. We also observe an initial large spike and a subsequent slight increase in the number of active processes.

Our emulation capability is a powerful building block towards understanding router malware. As a preview of the capabilities provided by RARE, we discuss how we assumed the botmaster role for two malware binaries at the end of section 2.3. Finally, we intend to make our tool, the malware binaries, and the data traces available upon request to researchers.

2.1 System Design and Implementation

We present an overview and highlight the novel capabilities of our approach.

Philosophy. We adopt the following approach: we start with a general purpose platform and learn what the malware needs to activate through a sequence of executions.

A visual depiction of RARE is shown in Figure 2.2. The key capabilities of the system are listed below: a) it can perform static analysis on the malware to extract information for its execution, b) it can instantiate an emulated router with hints on what

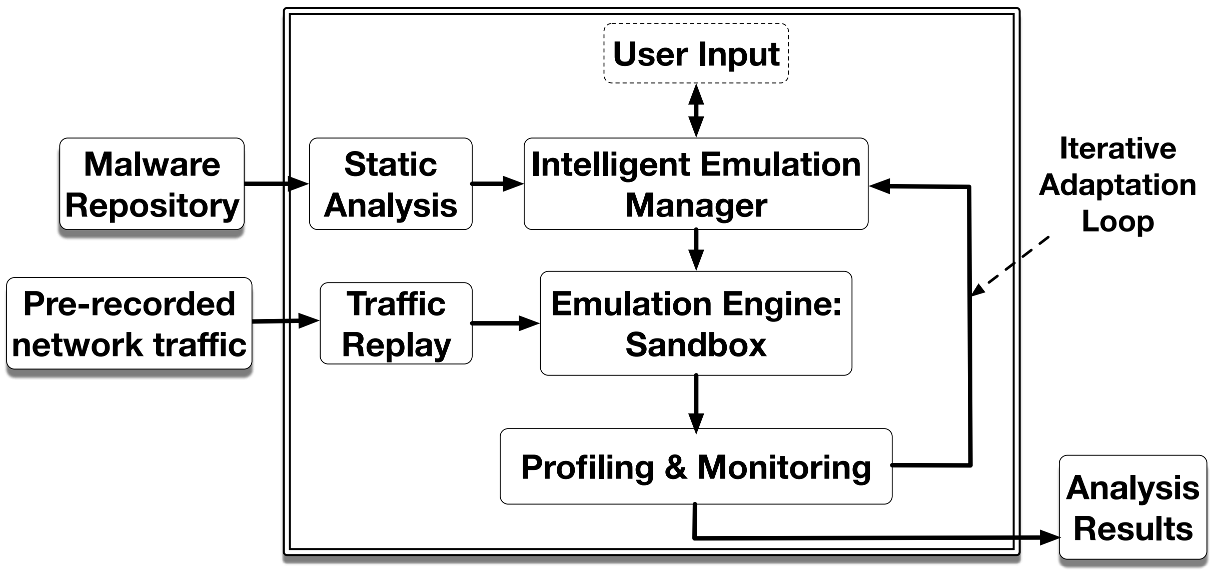

User Input
Malware Repository
Static Analysis
Intelligent Emulation Manager
Iterative Adaptation Loop
Pre-recorded network traffic
Traffic Replay
Emulation Engine: Sandbox
Profiling & Monitoring
Analysis Results

Figure 2.2: Overview of the key components of our approach. A key novelty is the feedback loop that enables previous runs of the malware to inform the subsequent run in order to fool the malware to activate.

configurations the malware wants to "see", c) it can replay arbitrary network traffic and response to malware requests, and d) it can monitor the malware and provide information to subsequent runs of the same malware. If the malware fails to activate, we repeat the process, and the last step provides information that guides the new execution.

Defining success: malware activation. We set as our goal for the emulation the activation of malware: we want the malware to feel "comfortable" within the emulator, so as to attempt to contact its bot-master. After reaching this point, we enter a new stage: if the C&C server responds, the malware will enter a stand by mode waiting for C&C commands. We define this stage as "activated" stage of the malware. A subsequent goal is to turn one's self into the bot-master, by reverse engineering the communication protocol with the bot, which we achieve for some of our binaries (see the discussion in section 2.3).

We present the key functionality, novelty and challenges of each module.

a. Static Analysis Module. The first step is to analyze a given malware binary statically to extract as much information as possible. The purpose is dual: (a) we want to inform the dynamic execution, and (b) we want to understand as much about the malware, which could have independent interest. Specifically, this module provides the following information to the dynamic execution: a) IP addresses and domains that malware will possibly contact, and b) files and resources that the malware will attempt to access. This information is collected by the Intelligent Execution module and is provided to the Emulation Engine module. In more detail, we currently use IDA-Pro [95] for the static analysis, but one could envision using other tools with similar capabilities. Therefore, we developed several non-trivial plug-ins, specifically for MIPS and ARM architectures, such as extensions to extract communication tokens, and control flow information which are the main components that assists this module.

b. Emulation Engine Module. This module provides the basic capabilities for emulating the router with the provided augmenting information and records execution traces. This module can: (a) instantiate an emulated router, (b) inject malware into the router, (c) replay pre-recorded network traffic or crafted C&C responses for the malware requests, (d) receive commands and information from the Intelligent Emulation malware in order to convince the malware to activate. The emulation is hosted in an Ubuntu server and `QEMU` [31] (an open source and widely used tool) is used to emulate the router hardware of interest, which here is `ARM` and `MIPS`.

We have made significant extensions and engineering in order to enable all the functionality. For example, we modify QEMU to recognize two subnets to represent the enterprise network and the rest of the Internet. We also added the ability to interact with the router through the network interfaces connecting to subnets, which is further discussed in the Traffic Replay module. For the firmware of the instatiated router, we use `OpenWrt` [7], which is fairly widely used codebase and it is considered as a reference firmware for routers. OpenWrt has the essential basic modules of a home router such as `DHCP` server, packet forwarding and routing, and a web interface. We added the monitoring capabilities to the instantiated router to collect execution traces for the router for further behavioral analysis, which we discuss in the Profiling and Monitoring module.

c. Intelligent Emulation Manager. The key novelty of RARE is represented by the Intelligent Emulation Manager module. This module drives the emulation in a way that achieves malware activation. First, it uses information from static and dynamic analysis in the previous two modules. The intelligence of the emulation is based on a feedback loop that gets execution information from each run. Every run is a clean start of the router augmented with information about the malware's requirement to fully execute itself. This loop represents the learning process of our approach. Second, the module facilitates the manual interaction with the malware, such as crafting C&C responses.

d. Traffic Replay Module. To observe a router in its natural element, we developed the Traffic Replay module, for replaying arbitrary network traffic to the router. Our module can get a real trace of two directions, incoming and outgoing, and replay it through the emulated router. This module is built on top of `tcpreplay` [21], but significantly we added new functionality. For example, the concept of incoming and outgoing traffic needed significant engineering, as it was not fully provided by tcpreplay. We also needed to take care of implementation issues, such as timing the replay traffic through the router. The network traffic we use is real traffic data from project *MAWI* [50], ensuring that we capture the beginning of each TCP flow across different days and different hours. Another key feature in this module is its ability to inject traffic at the command of the Intelligent Emulation Manager. This allows us to impersonate the C&C server by providing server responses and commands. In other words, our approach combines the knowledge of who the malware attempts to talk to, and what the malware expects to hear, through the deep profiling in the Static Analysis and Profiling and Monitoring modules, and we expect to be able to fully explore the intention and capability of the malware. We further detail the implementation of this module in Section 2.2.

e. Profiling and Monitoring Module. This module synthesizes information from the static analysis and dynamic executions with the following goals. We want to: (a) "understand" the malware in order to create an environment that will make it to activate, and (b) profile the behavior of the infected router in order to distinguish it from that of a benign one.

We list the types of data that this module collects and analyzes.

- Network Traffic: The module collects the network traces for both router interfaces (think incoming and outgoing) and analyzes them. The goal is to observe packets and flows generated by the malware, so that we can respond to them as if we are the C&C server. We can determine if a packet is generated by the malware by comparing the log files of the execution with and without the malware.

- OS System Calls: The module also collects the system calls at the operating system level. The goal is to extract information for augmenting the next iteration of the emulation. For example, the malware usually checks for existence of different files in the system as a means to infer which platform it is operating on. Another family of malware uses the */proc/sysinfo* file to infer the CPU architecture, while in another family they use `banner` file.

- System Processes: The module also monitors the processes in the router, which provides a complementary view on the behavior and intentions of the malware. For example, several malware binaries kill: (a) processes in the router in order to free up resources for themselves, (b) security processes, such as the `iptable` firewall process, and (c) user access processes, such as the `http server` process.

This module initiates the feedback-loop by providing the information to the Intelligent Emulation Manager to ensure that it can "fool" the malware to reveal itself and even believe that it is communicating with its C&C server.

Finally, this high-level description of our approach can be seen as a blueprint for a router malware analysis tool. Although in RARE, we have made specific implementation decisions for each module, functions and methodologies can be modified and replaced easily due to its modular design.

2.2 Replaying Traffic

In this section we show the technical implementation of the Traffic Replay module. We explore the steps that are taken to enable the network traffic replay.

Step 1. Prepare traffic. In our design of RARE our goal is to replay the previously recorded traffic as the live traffic that the simulated router processes. In doing so, given a `cap` network traffic file, we split it into two separate sets:

- Client traffic: We consider a given flow or traffic belongs to a client if they initiate the traffic: a TCP SYN packet was sent or in case of UDP, a DNS request was made. Also, we presume client traffic if ICMP port unreachable traffic is destined to the client.
- Server traffic: Following the same technique as above, we assume server traffic if the traffic is in a response to the clients such as sending TCP SYN/ACK or DNS reply.

Figure 2.3 shows by splitting the traffic into clients and server our emulation sandbox will perceive the two sides of a router which are the server side (known as `wan`) and the client side (known as `lan`). It is crucial to point that given the traffic is being replayed via a `bridge` network device by using Linux `TAP`, we have to rewrite the MAC address of the client and destination to the that of the `lan` and `wan` Ethernet address respectively. This is because in `TAP` setup, the routing occurs on the MAC layer.

2.3 Evaluation

We evaluate our method to assess the effectiveness of RARE in activating the malware and profiling the behavior of malware.

The malware binaries. In our evaluation, we use malware binaries that were collected from a community-based project, that requested to stay anonymous. These sources

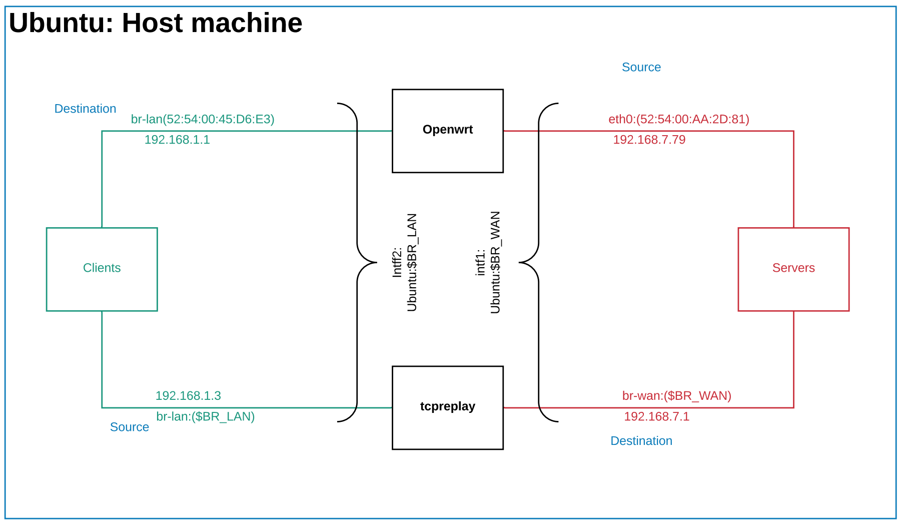


Figure 2.3: Overview of the client and server side of the sandbox in RARE's Traffic Replay module. Note that the MAC address of the packets have to be changed to the Ethernet address of the emulated router.

use honeypot and manual effort to collect these malware binaries. We have a total of 221 unique malware binaries targeting MIPS Big Endian (BE), MIPS Little Endian (LE), and ARM architectures. We focus on these architectures given that they are the most common ones in routers.

Network traffic. A key feature of our emulation is that we can do experiments using arbitrary network traffic. We report results with real network traffic from `MAWI` [51] with a duration of 1300 seconds. Note that we have experimented with other data traces, and also observed the router with no traffic at all.

The baseline approach. We define the baseline approach, as an emulation using the RARE infrastructure (OpenWrt over QEMU) but without any of the intelligence of our approach. We inject the malware in such an instance of an emulated router, and we monitor its behavior as the traffic passes through.

Table 2.1: The success ratio of our solution RARE compared to a baseline method.

		Baseline		RARE		
	Binaries	Raw No.	Percentage	Raw No.	Percentage	Avg Runs
All	221	**23**	**10.4%**	**208**	**94.1%**	3
Packed/Obfuscated	45	**5**	**11.1%**	**40**	**88.8%**	3
ARM	101	11	10.9%	91	90.1%	3
MIPS BE	77	6	7.8%	75	97.4%	2
MIPS LE	43	6	14%	42	97.6%	3

Evaluation.

For each malware and data trace, we compare the following approaches in executions with and without the malware: (a) Baseline run: which is the approach described above. (b) RARE run: The emulation gets augmented with the information extracted from the malware in each run. We define each execution as k^{th} run, being 1^{st} emulation informed by *Static Analysis* module and later runs informed by the behavior observed in the previous run.

Defining failure: no malware activation by the 6^{th} run. Setting a high standard for our approach, we require that we achieve malware activation by the sixth run. If this does not happen, we consider this a failure.

A. Evaluating RARE: 94% malware activation success. Using our tool we have been able to reach activation for 208 malware binaries out of 221. Table 2.1 details this

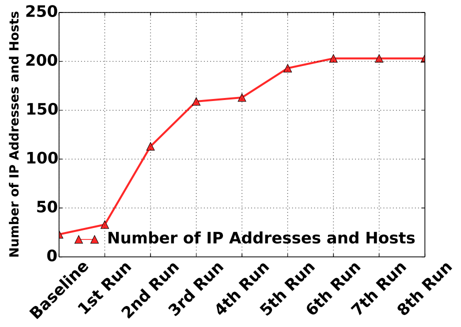


Figure 2.4: The number of extracted IP addresses and hosts rises with every run.

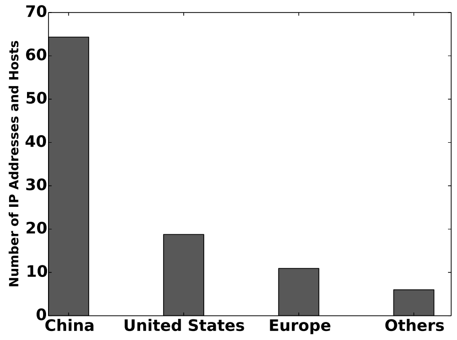


Figure 2.5: Distribution of the geolocation of the detected IP addresses.

success by comparing RARE with baseline approach. We report the average number of runs that RARE needed in order to reach activation for the binaries in that group that reach activation. We see that RARE requires a relatively low number of repetitions on average, namely three. We also separate the malware based on the target architecture: ARM, MIPS BE, and MIPS LE.

RARE exhibits great performance even for obfuscated malware. We compare the two approaches on 45 obfuscated malware binaries, as, for these binaries, static analysis cannot be used. Although there is a drop, our approach maintains a success ratio of 88.8%.

What does the malware need to activate? We observe an interesting and systematic progression in the types of requests that the malware generates. In the first few runs, the malware binaries typically attempt to change or check the existence of files, such as configuration files such as `/etc/ISP_name` or web interface files from *Linksys* and `TP-Link` routers, which are found specifically in home routers routers or specific brands of routers. Subsequently, some binaries attempt to tamper with different routing services, such as `libnss` and DNS, or change the routing table to subvert traffic. Typically, in the last run before the activation stage, the malware tries to resolve the hostname to locate its C&C server. In response, we inject DNS lookup responses using our Traffic Replay module. Note that, in our system, we retrieve the request that led to the failure using the Profiling and Monitoring module, we prepare for the subsequent run accordingly. As an example, when running one of the binaries an HTTP request to `userRpm/SoftwareUpgradeRpm.htm` (part of the `TP-Link` web interface) is observed which tampers with the installed firmware on the router. Failure to respond to this request will interrupt malware's execution. Using the Intelligent Emulation Manager module the appropriate web interface is installed for the next clean run of the router emulation.

Why do some malware fail to activate? This is an open question, which we will continue to investigate in our future work. In many of the malware requests listed above, we are able to provide a fake answer or a fake file. In most of the cases where we fail to do so, the malware tries to dynamically link to a custom library, which are not publicly available. Furthermore, these malware binaries are usually obfuscated, so faking the libraries is not straightforward.

B. Extracting useful information: 203 botnet IP addresses/hosts. We highlight initial elements of the information that we can extract with our approach. Overall, we find that RARE finds 203 malicious entities, IP addresses and host names, which are used used by the malware for botnet communications.

Static analysis: 48 addresses and hosts. Using static analysis, we traversed the CFG for all the paths from binary's *Entry Point* to system call `connect` and traced the input arguments values. We were able to extract 22 unique IP addresses and 26 host names.

RARE dynamic analysis: 155 addresses and hosts. Using RARE's consecutive runs, we extracted an additional 155 IP addresses and hosts. As shown in Figure 2.4 with every run the number of extracted IP addresses and hosts rises until the 6^{th} run after which no new IP or hosts are detected.

The dynamic analysis finds 75% of the malicious entities. The corollary

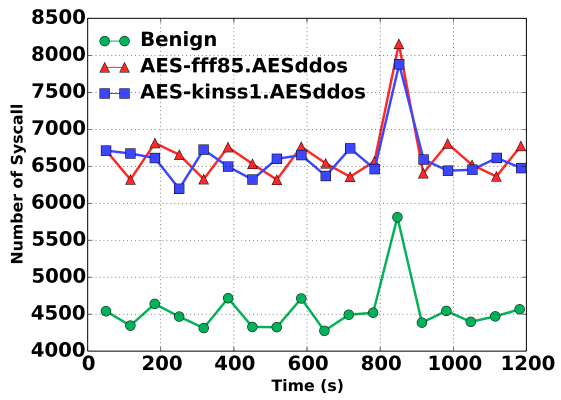


Figure 2.6: The number of system calls of a benign and two infected routers for the last 1200 seconds of the execution. The infected routers have consistently 1.5 times more system calls.

of the observations above is that the dynamic analysis is essential in detecting malicious entities of botnet communication. Using just static analysis, we find less than 25% of these C&C communication addresses and domain names. The issue is that the malware often obfuscates the addresses and the domain names, by using hexadecimal numbers and using number transformation and encryption techniques. For example, one binary has the following address generation approach: a hardcoded hexadecimal base value, and a function that adds decimals values to this base to obtain a series of IP addresses.

What is the geographical distribution of these malicious Internet entities? In Figure 2.5, we do a reverse look up to identify the geolocation of the IP addresses, and we observe that China (64.3%) and United States (18.7%) are the top destinations for hosting C&C servers. Many times these botnet entities could be compromised machines. In figure 2.4, we plot the number of malicious IP addresses extracted at each run of RARE. Initially, more information is extracted with each run, but this stops by the sixth or seventh execution

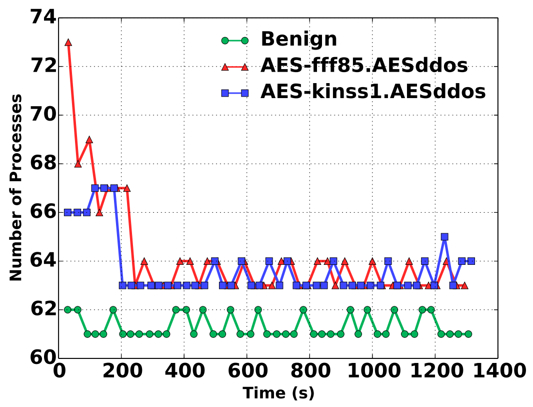


Figure 2.7: The number of active processes of a benign and two infected routers for our 1300 sec experiment. Initially, the malware spawns child processes to make itself persistent.

for most binaries.

C. Profiling infected router behavior. Our approach gives us the ability to compare the behavior of non-infected and infected router with a rich set of information at both the network and OS layers.

In all malware, we observe an increase in the number of system calls of almost 50% or more in an infected router compared to a benign one. Figure 2.1 shows a comparison of the number of system calls between infected router using the baseline, and the different RARE executions (numbers 1 to 8). This particular malware binary, reaches the activation stage at the third run. On the fourth run, we impersonate the C&C and we start issuing commands to the malware.

To better understand the malware, we show the number of system calls over time for two infected routers compared to a benign one in Figure 2.6. For visual clarity, we show only 1200 seconds of the execution to avoid the huge initial spike which corresponds typically to the reconnaissance of each malware. However, we do show the initial spike in the number of processes of an infected router in Figure 2.7: the malware makes itself persistent by spawning child processes.

Discussion: Becoming the botmaster. Using RARE we identify the functions available on two of the malware binaries from MIPS LE and MIPS BE. This was achieved by combining the static analysis and profiling information after the execution. We were able to convince the malware that we are the botmaster, and we were able to have it do: 1) HTTP flooding, 2) reverse shell, and 3) kill processes based on their process ID.

2.4 Related Work

We briefly review related work due to space limitations.

Emulation techniques: Several emulation techniques and tools exist, but they mostly focus on PC and Android platforms: `Anubis` [30], `PANDA` [68], `DECAF` [92]. The approach is to simulate the target platform and apply monitoring tools to record the execution traces of the malware. PC and Android platforms and malware differ significantly from router-specific ones, which is our focus here.

Vulnerability detection in embedded systems: Several recent studies focus on detecting vulnerabilities in the firmware of embedded devices [56, 46]. Chen et al. [46] argue that emulating platforms for specific firmware is not a trivial task since it includes emulating hardware components designed by vendors who do not necessarily practice a global Hardware/Software design standard. Other approaches [202] use real hardware to overcome this difficulty, but introduce the high cost and overhead. In our case, with thousands of router configurations, this approach would be very expensive and time consuming.

PC and smartphone malware studies: Many studies propose malware analysis tools using static or dynamic analysis. In static analysis, several studies focus on Control Flow Graphs characteristics [52, 105]. Static analysis on binary code requires platform specific tools, so PC-based or smartphone bases tools do not work for ARM and MIPs platforms. A limitation of the static analysis is that it does not work for obfuscated malware [133]. Several studies use dynamic analysis to classify and distinguish between different families of malware by studying the operation at the OS level [109, 115, 78]. In the PC and smartphone space, getting the malware to activate is an easier task given the more limited diversity in these platforms.

2.5 Conclusion

We propose, RARE, a comprehensive approach to analyze router malware. The key novelty is the augmented operation of our approach: instead of trying to guess the right router platform for each malware, we start with a generic one and we iteratively "adapt" it to fool the malware.

Our system provide the following key capabilities: a) perform static analysis on the malware, b) instantiate an emulated router, c) inject malware into the router and fool it to activate, d) replay arbitrary network traffic, and e) profile the malware behavior. Using real router malware, we are able to show that: (a) our system works effectively and manages to activate 94% of all our binaries, and (b) we can extract useful and insightful information from the execution of the malware. First, we find that we can identify malicious IP addresses and domain names, which subsequently could be investigated and blocked in firewall filters.

Second, we identify tell-tale signs of an infected router operation, such as 50% increase of the system calls.

Our approach is a solid first step towards developing a key capability for an under-served segment of devices. Although the results are already promising, we plan on expanding the capabilities significantly in two different dimensions. First, we will develop a more extensive static analysis capability, where we could infer the structure and key operations of the malware code. Second, we will further explore how to fully interact, and ultimately control both a bot, but ultimately a C&C server.

Chapter 3

Comprehensive Emulation and Dynamic Analysis Platform for IoT Malware

New tools and techniques will be required to combat IoT malware, since it is significantly different compared to computer-based and smartphone malware as discuss further in Section 3.1. The first key difference is that IoT malware can exploit a largely diverse set of devices, including industrial controllers, home sensors, and the power grid infrastructure. As a result, even finding which is the target device for a given malware binary is not an easy problem. Second, many of these IoT devices have limited resources, which changes the rules of the game: (a) the malware is designed to operate within these minimum resources, and (b) our ability to design counter measures is also limited. Finally, these devices have traditionally been built without extensive security considerations, and

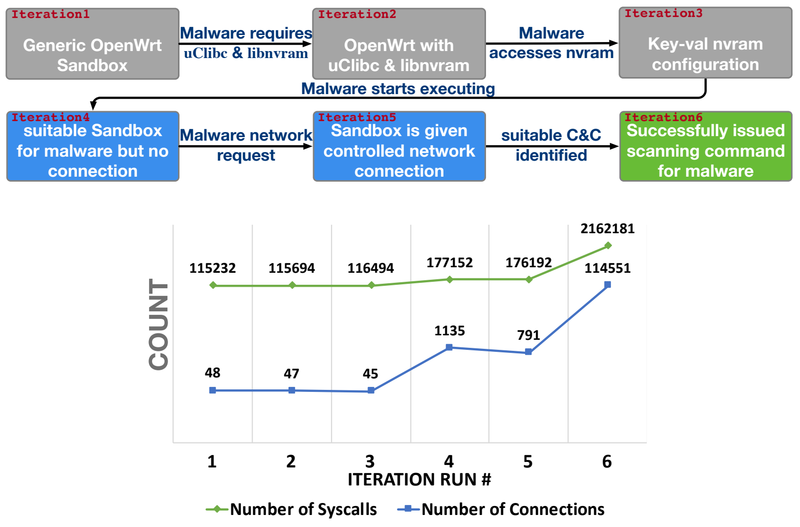


Figure 3.1: Our automated iterative approach is able to activate and impersonate the C&C server for an IoT malware.

this makes them a fertile ground for malware. It is indicative that, in most cases, IoT botnets are discovered after they deliver their attacks or infection [20].

How can we effectively analyze IoT malware and engage with it to perform an in-depth analysis? This is the problem we address in our work. We focus on conducting dynamic malware analysis in order to analyze its malicious behavior. Specifically, given an IoT malware binary, the goal is to: (a) activate the malware, by providing the appropriate target device configuration and (b) engage with the malware in order to explore as much of its functions as possible. We also consider a realistic assumption: we assume no prior knowledge about the malware binaries, which makes the problem harder, but also more relevant to practice.

Challenge 1. Activating the malware requires us to determine the right target device in order to emulate it. Each malware may expect device-specific files, such as drivers and libraries from a plethora of target IoT devices, and some binaries target very specific devices. In the absence of an appropriate configuration, the malware binary will terminate its execution.

Challenge 2. Reducing the manual effort in engaging with the malware is a challenge in studying its behavior effectively and at scale. Namely, even if the malware is activated, how can we see the full spectrum of its functions without requiring significant human effort and trial and error? Ideally, we would like to have the malware go through all its phases, including connecting to its botnet, proliferating, and attacking.

So far, there have been limited efforts that focus on dynamic analysis tools and emulation capabilities for IoT malware. A recent effort [57] analyzes Linux-based malware and its behavior focusing on the operating system level interactions. Other related work can be grouped in the following large categories: (i) Linux sandboxes and emulators [120] [65] [86] [161]; (ii) studies of malware traffic in the wild [43, 20, 119]; (iii) IoT binary and firmware analysis [46, 56]; (iv) Static and dynamic analysis of malware [115, 109, 53, 52]. We discuss related work in more detail in Section 3.4.

In this work, we develop RARE[1], a comprehensive emulation and dynamic analysis platform for IoT malware. Our approach is designed to addresses the two aforementioned challenges by employing two techniques that complement each other: (a) *Iterative Adaptation*, and (b) *Automated Interaction*. We depict the operation of our approach in Figure 3.1. After several failed attempts, we create the required target configuration that makes the malware activate in execution 4. From that point, we begin to "control" the malware by impersonating its C&C server. We elaborate on the key novelties of our approach next.

[1]Riverside's Augmented Router Emulator

Novelty 1. *Iterative Adaptation*: Activating the malware. RARE employs an *Iterative Adaptation* process, which incrementally creates the malware's target device configuration. The platform starts with a generic IoT hardware configuration, and keeps executing the malware, while fine-tuning the configuration. The number of iterations range from at least **3** and at most **8** depending on the type of malware, and whether it tries to narrowly target a particular device.

Novelty 2. *Automated Interaction*: Engaging with the malware. Complementing the iterative process, RARE has a suit of automation techniques that minimize the need for manual effort. We collectively refer to these techniques as the *Automated Interaction* process. This automation facilitates two distinct phases: (a) while generating the target device configuration, and (b) while impersonating its C&C server. In our platform, we have created a knowledge base of: (a) more than 5000 files, and (b) **148** unique C&C commands.

Our approach is an elegant and fundamentally different way to address the malware activation problem. The key advantages are: (a) **cost effectiveness:** we have one "adaptive" emulation instead of a large array of pre-made but fixed device configurations, and (b) **wide-behavior exploration:** our tool enables us to engage with the malware and observe its multi-faceted behavior.

We evaluate the effectiveness of our tool using **2885** malware binaries including **128** packed or obfuscated binaries, which we collected from public and private sources, as we discuss in Section 3.1. Our initial analysis results show the effectiveness of our platform in executing and analyzing IoT malware:

1. **Activation success: 93%.** We activate nearly all (93%) of our malware binaries. This includes all the 128 packed and obfuscated binaries.

2. **Activation of 173 hard to detect binaries.** We were able to activate **173** binaries, which other services and tools could not detect, even when provided with the binary. In fact, at the time of writing, we submitted these binaries to Virustotal [181], a point of reference in the security community, and none of the detection engines recognized the binaries as malicious.

3. **C&C impersonation success: 79%.** We impersonated the C&C server for **79%** of our binaries. We consider the impersonation successful if we can issue at least one command, that will make the malware initiate a behavior, such as its proliferation process, a DDoS attack, or self-termination.

Our work in perspective. Our work is a building block towards understanding, profiling, and mitigating malware outbreaks. Our dynamic analysis can provide: (a) "behavior signatures" and (b) strategies for counter-attacking the malware, e.g. providing instructions for taking over a botnet.

3.1 Platform Architecture

The system consists of several modules, which complement each other. We depict the system architecture in Figure 3.2 and we discuss the role and function of each module below.

The key novelty of RARE is two fold: (a) *Iterative Adaptation*, and (b) *Automated*

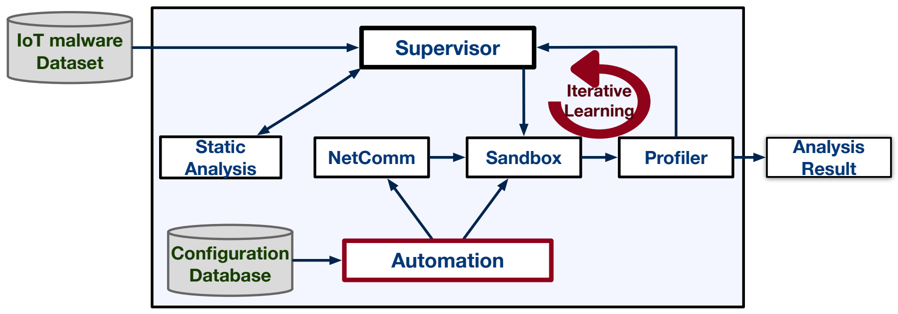


Figure 3.2: The architecture of RARE: 6 modules that work together to first activate the malware then interact with the use of *Iterative Learner* capability along with *Automation*.

Interaction. These capabilities work in a synergistic way to: (a) incrementally create the malware's target device configuration starting from a generic setup, and (b) minimize the need for any manual effort.

The *Supervisor* module: The *Supervisor* 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. 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. We use the term *iteration* to refer to each dynamic execution of the same binary. This module implements a key novelty of our approach: the *Iterative Adaptation* process, as we mentioned above and shown in figure 3.2.

The *Automation* module: This module automates the *Iterative Adaptation* operation and enables the *Automated Interaction* mechanism. It facilitates the analysis in two distinct phases: (a) while building the target device configuration, and (b) while impersonating the C&C server. For this reason, we have created two separate databases that provide: (i) a collection of commodity IoT configuration files, and (ii) C&C server protocol specifications.

One of the key challenges in designing this module is to understand the different type of resource requests and respond to them correctly. For example, if a malware attempts to accesses contents from non-volatile memory (`NVRAM`), this module will instantiate an NVRAM simulation.

The *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, which we can typically extract even for packed and obfuscated binaries. When the binary is neither packed nor obfuscated more information can be extracted. Leveraging IDAPro [95], we created significant new functionality using *IDAPython* for different CPU architecture to identify key functions, communication strings (e.g. embedded IP addresses), and in some cases, encryption keys and IRC channel login passwords.

The *Sandbox* module: This module is the work-horse of our platform. It is responsible for creating an emulated environment, where the malware is injected. The configuration is provided by the *Supervisor* module, as discussed above. Each execution starts without an infection, and the binary is injected once the emulation starts.

We highlight the key functions of this module. First, the module creates a virtual environment based on QEMU [31]. We modified QEMU to enable network interaction with the emulation instances. Second, this module adds monitoring capabilities at different layers of observation including: system calls, library calls, and network traffic. It records execution traces along with network traffic and feeds the generated log files to the *Profiler* module, as we discuss below.

The *NetComm* module: This module is responsible for all the network layer communication, essentially emulating “the Internet”. Once activated, the malware starts communicating: a) assessing the availability of connectivity, b) C&C server discovery and connection, c) proliferating, and d) perform malicious network attack. However, this poses a challenge of correctly identifying the intention of each packets that the malware generates.

The *Profiler* module: This module analyzes all the data that is collected during the emulation in order to: (a) understand deeply the behavior in the current execution, and (b) inform the next execution of the malware. The profiling synthesizes multiple levels of information including: system calls, file accesses, hardware peripherals accesses, and network layer communication. The analysis is performed by correlating the different level of traces through a dependency graph. For example, when analyzing the intention of each network packet, we use the series of system calls that lead to its creation starting from the creation of the process and including all the file system accesses.

The outcome of the profiling provides critical input to the *Supervisor* module, in order to set up the next execution. When analyzing a binary is completed, this module provides an extensive report about the a wide-range of behaviors of the malware to the user. The report can provide behavior signatures of the malware, which can be used to detect and contain malware infections.

3.2 Platform Implementation

This section provides a more in depth discussion and implementation issues of RARE. We describe the key novelties and highlight key points and challenges for each module and the key novelties: *Iterative Adaptation* and *Automated Interaction*.

3.2.1 The ***Supervisor*** module

The *Supervisor* module is the main component that drives RARE to perform the analysis.

The operation of this module starts with selection of a malware and the following steps are performed: (1) it sends the malware to the *Static Analysis* module for initial analysis, (2) the derived information from *Static Analysis* module will be used to start the *Sandbox* module for the first iteration of dynamic analysis, (3) later the *Sandbox* forwards the execution traces to the *Profiler* module for execution analysis, (4) the *Profiler* module generates analysis results, which include information about any failure in the dynamic execution, (5) the *Supervisor* module uses the derived information from the *Profiler* module to inform the subsequent iteration of execution, (6) with the use of *Iterative Adaptation*, the *Supervisor* module incrementally builds the target device for the malware. The goal is to drive the analysis towards "*activation*" and "*engaging*" with malware through C&C server *impersonation*.

The *Supervisor* module is also responsible for terminating the analysis of a binary. There are two reasons to do so:

1. *No further changes*: The analysis of a given binary should terminate, if we

cannot “invoke” new malware behaviors.

2. Exceeding the maximum number of iterations: This is a practical constraint to ensure that the emulations are not executing for an arbitrary amount of time.

3.2.2 The *Automation* module

One of the key novelties of RARE is the *Automation* module, which automates the *Iterative Adaptation* operation for the *Supervisor* and enables the *Automated Interaction*.

4.2.1 Automating *Iterative Adaptation*

The *Automation* module provides the configuration requirements for each iterative execution. This module has an extensive ***Configuration database*** which consists of repositories for device configuration files, libraries, drivers, and binaries. Currently there are more than 80 firmware collected from vendor’s website which includes routers and IP cameras. Our database consists of more than 5000 configuration files and binaries. We show case a subset of 10 firmwares in Table 3.1.

We define two types of resource requests: (a) software, and (b) hardware resources requests. Below, we discuss how the *Automation* module handles each type of resource request.

1. Software resources. In order to understand this resource request, we categorize the extracted data into four groups:

a. Lib: This group includes files found under known library folders such as `/lib` and `/usr/lib`.

	CPU	**Lib**	**Shell**	**LKM**	**Config**
Linksys	MIPS LE	45	38	4	5
Netgear	MIPS BE	154	334	27	36
	MIPS BE	73	253	33	347
Tplink	MIPS BE	63	13	58	19
	MIPS BE	55	18	64	20
	MIPS LE	46	68	18	26
Ubiquity	MIPS BE	260	937	290	49
	ARM LSB	43	54	6	18
DVR	ARM LSB	0	5	0	46
NAS	ARM LSB	939	577	N/A	423

Table 3.1: Overview of the files in our *Configuration database* for different architectures and vendor. Files are grouped into different categories: *library*, *shell binary*, *Loadable Kernel Module*, and *configuration files.*

b. Shell: This group includes binary files that are specific to command execution which are found in directories such as `/bin`.

c. LKM: This group consists of **L**oadable **K**ernel **M**odules, which are binaries used for loading kernel module into the operating system as device drivers, on demand. These are `relocatable` binaries with `.ko` extension.

d. Config: This group consists of all files found under different library directories, including `/etc`, `/mnt/Conf`, and other directories according to the firmware's setup.

Depending on the type, the *Automation* module searches for the particular file or binary in the repository and responds with the appropriate file(s). For example, when in an execution an embedded library like `libcms_msg.so` is requested, the response includes a copy of the library along with its dynamically linked library `libc.so.0` and the corresponding C library.

2. Hardware resource. This includes any hardware peripheral resources that may be accessed during an execution. In this situation, we consider two cases of requests: (a) reading configuration setting, and (b) modifying the hardware setup.

a. Reading: The module creates a key-value based type of driver that will return values that indicate success. For example when the `NVRAM` is accessed to read a hardware configuration, a key-value service will return a string that indicates success.

b. Writing: The *Automation* module manages this access the same as "reading" a configuration. For example, there are cases where the malware binary modifies the `watchdog` timer to prevent the device from rebooting. We understand the possibility that a successful and realistic hardware configuration modification is part of the observable behavior of malware binary. However in the current implementation, RARE does not emulate the real hardware modification. Simulating embedded devices hardware and driver is a known issue and several work [202] has been proposed, however such solution is not practical when analyzing binaries in scale.

What happens when the requested resource is not in the database? First, we log this request which gets provided as input for improving the next execution. Second, we respond with a *"dummy"* file based on the type of the resource (e.g. shell binary or configuration). Interestingly, this can be sufficient for the malware to continue, if the malware wanted to verify the existence of such a file, or it does not use its content in a substantial way. For example, one of the binaries reads the content of the file `/etc/words` to generate `nickname` for joining the IRC channel: any non-empty text dummy file would satisfy the request.

Clearly, as we add more information to our *Configuration database*, the more we improve the capabilities of our *Automated Interaction* process. As we will see later, even our current initial database is quite powerful.

4.2.2 Automating the C&C Impersonation

As shown in Figure 3.1, after activating the malware binary, we want to impersonate its C&C server. The challenge is to identify the connection attempt packets and the C&C communication protocol. We were able to find some of the IoT malware source codes [18, 57], which we use in our source code repository, as we discussed previously.

We studied **174** IoT malware source code and created **5** C&C server simulations, which we incorporate into RARE. Table 3.2 shows the type of the C&C communication, number of DDoS function commands, and other commands such as reverse shell or killing the bot. As shown in this table, in two cases killing the bot requires a **passphrase**.

The *Automation* module selects the appropriate C&C server communication protocol, which we briefly explain below.

a. Extracting information from the packets. We parse the packet that the malware sends to the server, and extract revealing keywords. For example, if the packets include keywords related to `IRC`, we use the IRC C&C protocol. We have created a "dictionary" of keywords that are indicative of different types of C&C protocols by examining

C&C name	**C&C type**	**# of DoS funs**	**Other cmd**
Lizkebab	*Plain text*	4	Shell, kill
Gafgyt	*IRC*	7	Shell, Kill with pass
Kaiten	*IRC*	15	Shell, kill with pass
Remaiten	*IRC*	4	Shell, kill
Mirai	*Plain text*	4	Stop scanning

Table 3.2: Given the **174** source code, we simulate C&C servers with DDoS commands and any other command such as enabling reverse shell or killing the bot.

the malware source codes in our repository.

b. Trying all possible servers. If we cannot identify a target server, we will iterate through all our possible C&C protocols.

c. Creating an "echo" response. In case, none of the previous steps works, we "echo" the packet that we received. Note that this is not considered a successful C&C impersonation even if the malware exhibits different behavior upon receiving the response.

3.2.3 The *Static Analysis* module

We use static analysis to guide and facilitate the dynamic analysis, which is our focus here. The first analysis on a malware binary is performed by *Static Analysis* module. On the one hand statically analyzing binaries can help understanding the binaries. On the other hand, it is commonly known that malware developers make use of runtime packers to obfuscate the binary in order to evade common static analysis techniques by changing structure of the binary. We use this module to extract as much information we can to facilitate the analysis process. *Static Analysis* module performs two types of analysis:

1. ELF analysis: By using `readelf`[80] we analyze different sections of each ELF file and identify used CPU architecture, and information about dynamically linked libraries. RARE uses this data to select the emulation's CPU architecture and dynamically linked libraries.

2. Structure 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, however we also adapt established techniques in analyzing the structure of the binary [78, 62]. In more detail, we perform control flow analysis to extract artifacts such as: (a) embedded IP addresses and hostnames, (b) encryption keys in `Mirai` instances, (c) channel names in IRC based botnets, and (d) **passphrases** used in the IRC channels. In the current implementation, we use IDA Pro [95], but other disassemblers can be used, such as Ghidra[135]. At the time of writing, IDAPro is the most effective disassembler [17]) and with the use of *IDAPython* we employ the control flow analysis and guided symbolic execution for ARM and MIPS architecture.

The extracted information will be provided to the *Supervisor* module to initiate the dynamic analysis through the *Sandbox*.

3.2.4 The *Sandbox* module

The main analysis of a binary happens in this module. It enables the fundamental platform for dynamically analyzing a malware binary. It creates a fully functional emulated IoT device based on the configuration instruction that it receives from the *Supervisor* module. In each execution, the *Sandbox* is configured to setup an instance through the following

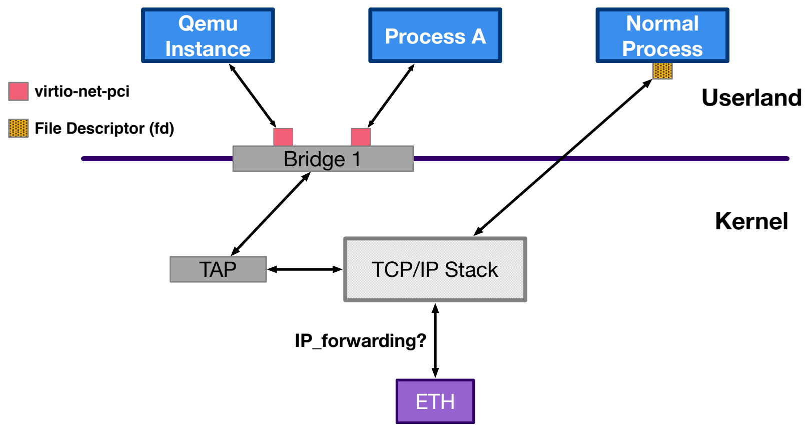


Figure 3.3: Overview of using TAP/TUN in our work to

step:

1. QEMU. We used QEMU [31], an open source project for CPU emulation. We modified QEMU (added/modified ˜2000 LOC), to introduce the networking and monitoring capabilities required to engage in communication with the *Sandbox* as we explain later in the Networking capability part of sandbox. Overview of the intended architecture for the QEMU is shown in Figure 3.3.

2. Filesystem and configuration resource setup. The filesystem includes required files and binaries that are described by the *Supervisor* module for the dynamic execution. As we explained in Section 3.2, the *Sandbox* module requests configuration resources from the *Automation* module. A configuration resource can include: (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, (c) device configuration files for example `/mnt/Config/` in *ZyXel* devices.

3. Configuring the kernel. We used the Linux kernel based on *OpenWrt* [7]. The initial standard C library is *glibc* and depending on the resource requirement, *Sandbox* can select different C library such as `musl` or `uClibc`. We create a modified kernel that supports: (a) system call monitor, and (b) hardware peripheral emulation.

a. System call monitor: We collect system calls though kernel with the use of kernel probes (`kprobe`). We use `kprobe` over userspace system call monitoring tools, such as `strace`, to prevent the malware binaries from detecting the monitoring capability. As we will ass in Section 3.3, there are binaries that abort execution, once they detect that `strace` is "attached" to the process.

b. Hardware peripheral emulation: The goal is to intercept system calls and requests to hardware peripherals to emulate them. This includes requests to non-volatile memory (NVRAM) and non-standard `IOTCL` system calls. We implemented a specific `libnvram.so` that works in a key-value fashion based on the contents from a `nvram.conf` file. Later we populate the `nvram.conf` file based on the requests to the NVRAM.

4. Injecting the malware. After setting up the file system and kernel modules, we place the malware sample in the `/root` directory. When the instance starts running, the `init` process will execute the binary with `root` user privilege.

5. Networking capability. Setting up and emulating a network infrastructure is critical in creating a realistic emulation. For example, some of the binaries use specific network interfaces chip(NIC) for communication such as `hwsim0` and `wlan0`.

Moreover, in analyzing malware binaries it is necessary to contain the network

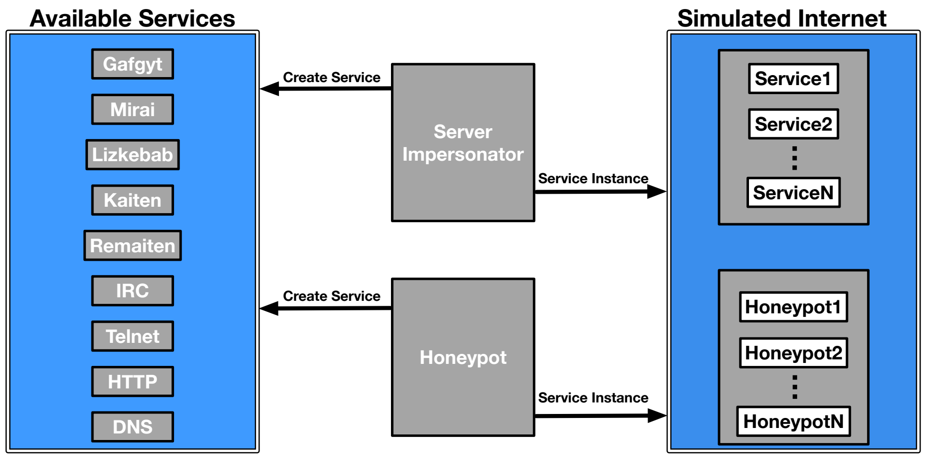


Figure 3.4: Architecture of the *NetComm* module.

access and prevent any threats such as attacking the experimental infrastructure. We leverage the network `TAP` option to enable network accessibility. Each instance will be assigned an IP address (by the `dnsmasq` service) to enable interaction with the *NetComm* module which we discuss in Section 3.2.5. In Figure 3.4

6. Execution trace. In each execution, the *Sandbox* module collects execution traces at the OS and network layers. As previously discussed, we use `kprobe` to collect the system calls. We directly collect data from QEMU. We do not rely on `tcpdump` inside the emulated device, since there are malware binaries that terminate any monitoring process upon activation. The collected execution traces are then sent to the *Profiler* module for further analysis as we discuss in Section 3.2.6.

3.2.5 The *NetComm* module

One of the key capabilities of this platform is the *NetComm*, which provides network communications for the *Sandbox*. As discussed previously, this is a crucial step in exploring malware's behavior in level of communication. A malware may use a network connection for different purposes. In this work, we consider the following three common network behavior in malware communication: (a) the malware checks for the network availability with active services, such as DNS, (b) it contacts the C&C server, and (c) it starts scanning IP addresses or port numbers in its proliferation phase.

a. Active network services. The goal here is to make the binary to believe there is network access availability. In doing so, the *NetComm* module uses information derived from previous execution iterations to determine: (a) the contacted server IP address or hostname along with the port number, (b) the transport layer protocol of choice. Using this information, *NetComm* creates instances of *Server Impersonator* to intercept the requests and engage with the malware binary.

The main challenge is to correctly identify the communication protocol used. On the one hand there are common communication purposes, such as DNS lookup using Google's `8.8.8.8:53`, while on the other hand it is important to not assume the intention of the connection based on the communication end point. For example a sophisticated malware can use DNS tunneling to obscure its communicate with the C&C server [66].

b. Server Impersonation. The other key goal is to impersonate[2] the C&C server to which the malware tries to connect. As discussed in Section 3.2, we have developed an automated capability to issue valid C&C commands to the malware binary and make the malware take action in response. This helps us reveal interesting malware behaviors: such as ordering the malware to enter its proliferation phase or terminate itself, as we see

[2]Our goal here is not to reverse engineer the communication protocol fully, which is a research topic in its own right [40, 60, 41] and part of future plans.

later in Section 3.3.

c. Honeypot. The goal here is to detect the proliferation techniques of the malware. 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 type of packets.

Distinguishing the intent of a connection. A key challenge is to differentiate between scanning traffic and a C&C discovery effort. For example, there are binaries that scan randomly generated IP addresses on port `23` and at the same time attempts to connect to their C&C server on the same port number. Therefore, redirecting all traffic on port `23` to the Honeypot will interfere with the effectiveness of our C&C impersonation.

We overcome this challenge by integrating a *Honeywall* technique [174] to redirect the C&C server traffic based on its IP address to the *Server Impersonator* part of the *NetComm*. We identify the C&C server 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 are, the more likely that the destination is a C&C server [186].

In our current implementation, we use `Cowrie` [145] for `SSH`, `Telnet` and a simple HTTP server for web interface and `UPnP` types of attacks. Based on the scanner's port number, we create instances of Honeypot listening to the particular port. For example, if the scanning is on port *7457* (`UPnP` service), we create an HTTP server honeypot on the particular port number.

In Figure 3.5, we show the interaction between the *Automation* module with

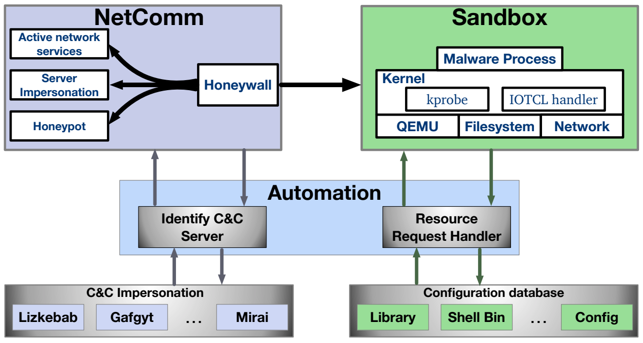


Figure 3.5: The *Automation* module in action.

the *Sandbox* and *NetComm* modules. In each *Iterative Adaptation* process, there are *4* steps(shown with the arrows) that each module has to take in order to request resource or identify C&C server for impersonation.

3.2.6 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 we monitor the malware at both the OS and networking layers and we **combine** the information for a deeper understanding of the intention of each malware action.

A. OS layer analysis: The module analyzes the system call of the the whole emulated system's and the goal is to identify and focus on the system calls that generated by the malware. To do this, we monitor the system calls by each process and focus on malicious processes: which are processes created by other malicious processes. In this analysis we employ a careful data dependency analysis proposed in [109] on the system call traces. This careful analysis helps us discover the root cause of error and failure in sample's execution in addition to more in depth analysis.

Following we describe each of these groups and our technique in identifying the errors and failures.

1. File access monitoring: We examine the trace for file related system calls such as `open` and `fstat` to determine the type of access (read/write) intended for the file as well as the returned result. This helps us identify the missing files or binaries, which often lead to the malware dying. The missing resource will be reported to the *Automation*, which will try to provide it in the next execution.

2. Network-related system calls: We also monitor carefully networking system calls such as `connect` and `sendto`. We monitor which process generates the system call and data dependencies, such as what information was included in a particular packet.

Note that some network system calls reveal activities that do not generate packets, and thus, we would not have observed them at the traffic trace. For example, some binaries: (a) create packets that are crafted for non-existence network interfaces, or (b) create raw sockets on specific network interfaces to sniff the the traffic.

B. Network traffic monitoring: We monitor the network traffic by collecting all the PCAP files through QEMU's network interface. As we mention earlier, we combine this information with information about networking system calls, which helps associating network packets and flows with system processes. This is critical for identifying malware generated packets. Analyzing the network traffic in depth is critical for impersonating the C&C server effectively.

As the goal here is to enable the malware to activate and engage, we focus on the cases where we fail to do so, and we summarize below.

I. Software resource failure: This refers to any failure due to software operation. More specifically if the malware fails to access configuration files, library and shell binaries, or kernel modules this is treated as a failure that can be resolved by making the particular file or binary available in the later execution.

II. Hardware resource failure: There are situations that the malware fails to access certain hardware peripherals (for example NVRAM). This stands as a more challenging resource request for the *Sandbox* module, since emulating hardware is not as trivial.

III. Communication failures: This describes the case where the malware is not receiving the communication response that will encourage to not only not terminate itself, but to exhibit its full behavior.

IV. Trigger checks: As discussed in earlier, the malware often attempts to detect an emulation environment, and and if does, it shuts down. Our goal is to understand what are the triggers for each malware, which can help both other emulation studies, but also develop techniques to contain the malware spread. For example, if the existence of a file triggers a shut-down, making devices have that file could protect them from that malware.

The malware profile and the failure-related information is provided to the *Supervisor* module, which will drive the analysis in subsequent executions for a more in-depth understanding of the malware binary.

3.3 Evaluating RIoTMAN

In this section, we evaluate the effectiveness of RARE using our malware binaries.

Implementation and safety issues. We implemented RARE on an *Ubuntu 18.10* server in a lab machine with 16 GB RAM memory and 1TB hard drive, and Intel Xeon E5 CPU. For safety, the server is in a private network and isolated from other campus networks and the Internet.

Conducting experiments: details and logistics. We provide some practical issues on how the platform will be used.

a. Clean Starts: Each malware execution starts with no prior infection in the emulated device.

b. Generic platform: Each analysis starts with a generic configuration which the *Iterative Adaptation* process will incrementally modify to build the target device. The generic configuration is based on a simple OpenWrt as the embedded OS with `glibc`. We use QEMU as the CPU emulation. No further hardware peripherals, or network accessibility and services are provided in the first iteration.

c. Infection Process: The infection can take place at a default time and fashion, but we also allow the user to customize these choices. By default, malware is placed in the `/root/` directory and we start the malware through the start up process (`init`) in `root` privilege mode. Although one could start the malware in user mode, our goal is to give the malware root privilege to avoid any access issue.

d. Experiment Duration: By default, in each execution, we observe 20 minutes of malware activity, unless otherwise stated. Naturally, the duration is a changeable parameter, but we found that 20 minutes is a sufficient amount of time. We noticed that in three cases where the malware suspends its execution via a sleep command, which can be used as a technique to avoid automatic analysis tool.

e. Activation Success: To have a clear definition for success, we consider that the malware activates when it initiates its first communication packet. Our rationale is that the malware observes it is in the "intended" target environment: a) it has identified the emulated device as appropriate, and b) it begins the first step to achieve its mission: connect with the botnet.

3.3.1 Activation effectiveness of RARE

Key Result 1: We activate 93% malware binaries. We argue that our iterative approach works: it is feasible to start with a generic configuration and morph it into the desired target device. Out of the 2885 binaries, we are able to activate 2884 binaries. The number of iterations range from at least **3** and at most **8** for a successful activation.

We investigated the reasons why we could not activate that last binary. Using manual inspection, we found that the binary requests to access resources from a hardware peripheral specific to a *NETGEAR* home router. The shortcoming stems from a set of configuration values that are initialized in the boot up phase in the actual *NETGEAR* device, and it was not straightforward to replicate.

Key Result 2: We activate 173 undetected malicious binaries. We were able to activate and observe the malicious behavior of 173 binaries, which Virustotal failed to report as malicious. At the time of writing this paper, we submitted the instances and the returned results shows that the binaries remained as undetected.

Are these binaries really malicious? As we will show in Chapter 4, we impersonate the C&C server for 131 (˜76%) instances out of the 173 undetected binaries. This result confirms the maliciousness of the majority of these binaries. Furthermore, manual inspection suggests that the remaining binaries are also malicious.

Why does Virustotal fail? We examined the characteristics of the unreported 173 binaries to understand the shortcomings in identifying them as malicious binaries. Below, we describe some initial observations as to the challenges that these binaries impose, and we highlight how our approach was able to overcome them.

1. Packers and obfuscation: We identified 12 binaries among these 173, which use `UPX` and other non-generic runtime packers to bypass static detection signatures. In another instance we observe the malware binary uses obfuscation technique to scramble the data part of the binary. In 7 instances based on *Mirai* the malware binary uses encryption key or series of keys such as `0xBAADF00D` and `0x33C001DE` as opposed to the original key observed in the source code(`oxDEADBEEF`) [19].

2. Dynamic linked libraries: There are malware binaries that rely on custom libraries that are found in the firmware of specific devices, such as `libacos_shared.so` and others. The absence of these custom library files leads to failure to activate. In RARE, we are able to overcome this through the combination of the *Iterative Adaptation* and *Automated Interaction* operations.

3. Non-DDoS malware: In some cases, the malware does not exhibit botnet behavior and instead they manifest "passive" functionalities, such as: (a) setting up network traffic sniffer, or (b) creating a reverse shell on specific port number.

3.4 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 [120], Detux [65], and cuckoo [86], 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 [57], 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 [62], focuses on routers, a subset of IoT devices, and does not provide any automated engagement capability as we do here.

To the best of our knowledge, none of the above approaches combines all three elements: (a) building the IoT device configuration iteratively, (b) providing automated capability to communicate with the malware, and (c) profiling the malware at the network layer.

b. Measurement studies of malware traffic in the wild: Several studies analyze the large scale effects of real attacks using observed data [119, 73, 114, 118, 117]. A recent study studies the Mirai botnet, which was enabled by IoT malware [20]. 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 [138, 34, 15, 47] In addition, other efforts focus on the privacy concerns introduced by such devices [157, 8]. 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 [56, 197, 76] and dynamic analysis [46]. 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 [115, 109, 110, 106]. 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 [133]. Several efforts develop techniques to understand the structure of the binary [16, 52]. Other efforts use symbolic execution to explore the execution path of binaries, such as project `angr`[170].

3.5 Discussion and Future Work

We discuss the usefulness and limitations of our approach.

a. Do we need IoT-specific malware tools? Throughout this work, we saw the need for IoT-specific approaches and tools. First, we have the challenge of identifying the target device configuration. This forced us to create the IoT-specific *Configuration database* as we discussed earlier. Second, IoT malware targets different architectures than PC-based malware, and some of the OS-level tools require significant adjustments (see Section 3.2). Further, we identified distinct IoT requirements and behaviors, which point to the need for novel methods.

b. 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 [72], (b) Virustotal corroborates that our binaries are indeed IoT malware, and it spans at least 20 malware families, as we have mentioned earlier.

c. Will RARE work with new malware? Clearly, we cannot guarantee that RARE 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 powerful as information is added in its knowledge base. Furthermore, the 93% success rate with our DNew dataset is a promising indication.

d. Can we fully explore the malware behavior? This is a challenging research question for any dynamic analysis approach. We are, arguably, the first approach to attempt an automated malware engagement capability at the network layer. Automating the server impersonation is an essential step. We saw that RARE is capable of issuing at least one successful command for 79% of the binaries. We intend to further improve this capability in the future in an effort to observe as many as possible of the phases of the malware life-cycle. Note that exploring the execution paths of a binary is a hard problem, which requires a combination of static and dynamic analysis [201, 170].

e. Is an adaptive configuration creation necessary? We argue that it is, and at the very least, it is more elegant and efficient. In traditional sandbox techniques, the analysis is done via pre-configured emulators [57] 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 possibly less successful. As we showed in Section 3.3, 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.

3.6 Conclusion

Our approach is an elegant and conceptually 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 Interaction*. 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 us understand the intention behind the malware activity. 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 RARE as an important step towards an effective dynamic analysis capability for IoT malware.

Chapter 4

A Longitudinal Study of IoT Malware Behavior

With the rise of Internet of Things devices in our daily lives and the lack of security in their implementation [55], more devices are susceptible to become part of a malicious botnet. Firstly, a botnet of IoT devices that were infected by Mirai malware have caused a world record DoS attack [20]. Secondly, there is widely-available source code of IoT malware, such as *Mirai* [19] and *Lightaidra* [18] making it easy for black hat hackers to create their own botnet. Lastly, there is evidence of IoT malware getting better: new families appear and existing families evolve and adopt sophisticated techniques, including proliferation techniques, and types of C&C discovery mechanisms [191].

The problem: How can we study the characteristics of an IoT botnet? This is the problem we focus on in this work. We conduct multiple iterations of dynamic analysis on IoT malware binaries and interact with them on the networking level to collect network traffic behavior. More specifically, given an IoT malware our goal is to: (a) analyze the techniques used for malware proliferation, (b) understand how the malware finds and connects with its C&C server, and (c) identify the communication protocol used to operated the malware and trigger its different functions.

Analyzing the networking behavior of a given malware is faced with three main challenges: (1) an analysis environment to install and collect networking behavior of IoT malware, (2) identify the type and the intention of traffic generated by the malware, and (3) by replicating the communication protocol, operate the malware to trigger its different functionality.

Previous work: There has been limited work focusing on analyzing IoT malware traffic, especially their C&C traffic. Previous work can be summarized into the following categories: (i) Analyzing IoT malware [65, 57, 120]; (ii) Static analysis of malware analysis [52, 124, 132, 163, 16]; (iii) Dynamic analysis of non-IoT malware [115, 109, 110, 106]; (iv) malware network traffic analysis [119, 73, 114, 118, 117, 97, 190, 84, 85, 37, 54, 83, 66]. We explore these and additional related work in Section 4.3.

In this work, we conduct an extensive and large-scale study on IoT malware communication patterns. First, we develop a non-trivial capability to engage with a malware binary by facilitating/automating the impersonation of its bot master. This impersonation combines: (a) the identification of the right communication protocol, and (b) issuing specific botnet commands. Second, using this capability, we conduct large-scale study on 2885 malware binaries caught in the wild and spanning roughly 21 families. We are able to impersonate the "botnet" for 25% of the families (78% of the binaries) and get them to enter different phases such as proliferation, and attack phase. Note that the binaries within each family still exhibit variations.

Specifically, we focus on the networking behavior of the IoT malware which includes the traffic from proliferation phase and the malware C&C operation. In Figure 4.1 shows the architecture of our work which consists of an analysis environment along with a server impersonation capability for IoT malware.

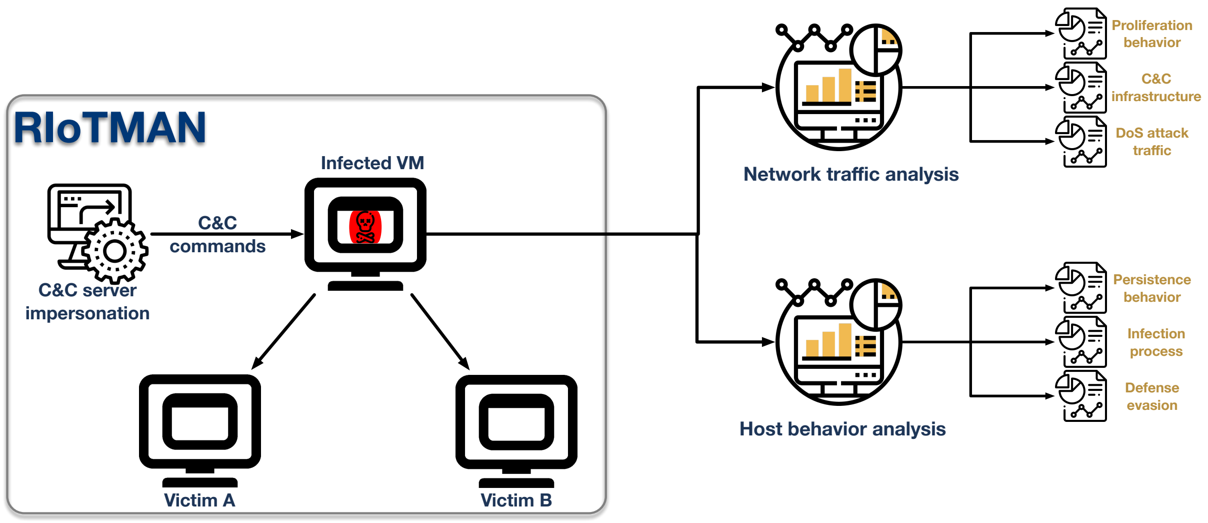


Figure 4.1: In this work, we extensively examine the networking and host behavior of all the IoT malware.

We conduct extensive dynamic analysis experiments on 2885 malware binaries collected from the wild, using a RIoTMAN from Chapter 3 to enable a large scale study as well as having the capability to operate the malware. We summarize some of the important findings of this study:

1. Distinct host scanning defines botnet goal. We identified 2 unique scanning techniques which attribute to the different characteristics of IoT botnets; we find

Mirai malware is specifically designed to recruit more devices. While in other cases, e.g. in IoTReaper and Hajime malware, more diverse set of devices were infected via exploiting at least 19 unique firmware vulnerabilities.

2. IoT malware proliferation requires a loader server. In their proliferation phase, IoT malware is *required* to have a server where it hosts the same malware but created for different CPU architectures such as MIPS and ARM. This is because IoT devices are a diverse set of devices built on diverse set of CPU architectures.

3. IoT malware uses one or many C&C infrastructures techniques. We observe multiple techniques used to connect to Command and Controller(C&C) server and/or join botnet. In some cases a combination of multiple techniques is used. E.g. there are malware that use Domain Generation Algorithm (DGA) along with fixed domain names. Recognizing the C&C infrastructure is especially important to stop the operation of the botnet.

4. 5 major unique protocols used for operating majority of IoT malware. Although there are more than 20 IoT malware families identified, most of them use one of the 5 C&C communication protocols. Given some of the IoT malware are based on previous open source bots (Section 4.1), we are able to impersonate the C&C server for 78% of all the malware binaries. We are able to operate the malware to start DoS attacks, reconfigure itself or terminate itself. We observe a total of more than 50 unique DoS attacks.

Our work sheds light onto the unknowns of IoT botnets and investigates the different techniques that they use. Our goal is to release the dataset collected from these experiments for the community to further the research in this field and bridge the gap between the current defense mechanism and the mechanism required to stop IoT attacks.

4.1 Experimental setup and Methodology

In this section we describe the experimental setup as well as the methodology used to perform malware analysis in large scale.

4.1.1 Experimental setup

Given the main goal of this study is to analyze the networking behavior of IoT malware, the goal is to have an analysis environment that can support: (1) the analysis environment should install and execute any IoT malware, (2) to avoid further malware proliferation, the analysis environment should not be connected to any network, and (3) a semi-realistic network connectivity should be made available for the malware to interact with the environment. Aside from the points mentioned, it is crucial to have the analysis done in an automated and accurate way to enable analysis at scale.

In this project we used RIoTMAN from Chapter 3. We added network interaction capability to provides network communications for the analysis environment which is a crucial step in exploring the behavior of malware. Here, we support the following two main types of communication:

a. Intercepting proliferation traffic. Inspired by honeypot project [174], we redirect the proliferation traffic to instances that simulate services that are targeted during this phase. In this work, we use `Cowrie` [145] for `SSH`, `Telnet` scanners, and a simple HTTP server for attacks targeting web UI and *UPnP* interfaces.

b. Impersonating the C&C Server. As previously discussed, our goal is to impersonate[1] the C&C server for the under study malware to trigger its different functions. This helps us reveal interesting malware behaviors: we can make the malware enter its proliferation phase or terminate itself. We show an overview of the experimental architecture in Figure 4.2.

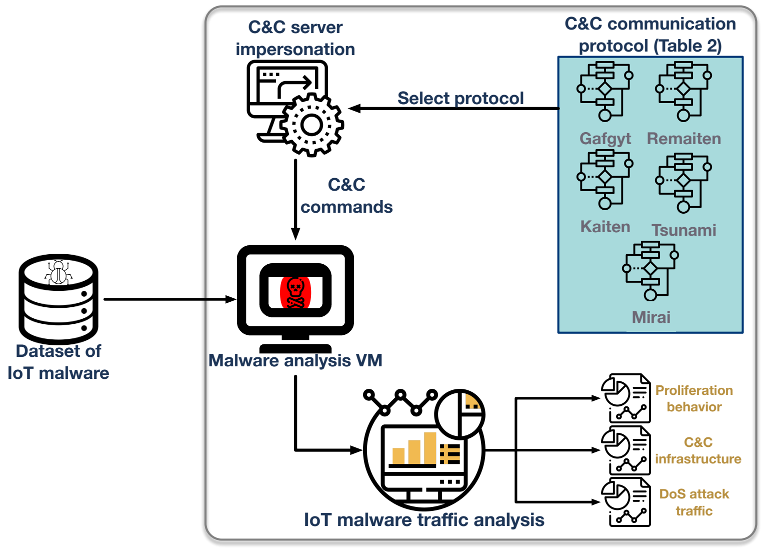


Figure 4.2: We use IoT C&C server communication protocols (Table 4.3) to operate malware in the analysis environment.

4.1.2 Methodology

The goal is to distinguish the different types of communication network traffic that are generated in our confined environment into three separate groups: (1) Proliferation traffic: this group of communication is used to expand the botnet by recruiting other devices, (2) Control traffic: this is any traffic generated between the malware and a centralized

[1]Our goal here is not reverse engineering the communication protocol, which is a research topic in its own right [40, 60, 41].

Command and Control (C&C) server or peer-to-peer network to direct the botnet, and (3) Attack traffic: this refers to the group of network packets that are generated to carry out attacks such as volumetric network packet flooding. Other network communication traffic generated by the malware such as checking internet availability is out of scope of this study and are treated as auxiliary traffic.

To differentiate between the types of communication, we use heuristics developed using domain expertise and empirical observation. More specifically, we manually inspected D_{train} =500 binaries to derive the following intuitions in identifying the type of network traffic:

1. Proliferation traffic: Here, the malware contacts different IP addresses on one or multiple port numbers. We identify this group of communication by redirecting the traffic to honeypots. When the malware is able to make a connection to a particular port number, it performs a brute force attack or sends an exploit payload to the destination.

2. Server Discovery: All malware binaries want to communicate with a centralized C&C server or join a peer-to-peer network to receive commands from the botnet operator. Upon establishing foothold on the infected device, malware tries to contact one or multiple IP addresses or resolving domain name. In case of connection absence, malware periodically fails to communicate with the intended destination. However, when there is a connection, it tries to maintain the communication by sending data.

3. DoS attack traffic: Given their nature of volumetric scale, DoS traffic in our confined environment can be identified with a heuristic that will get triggered if any of the following meets: i) more than 50 packets is sent on same destination IP address in a

Table 4.1: Evaluation of our heuristics for characterizing communication intent on D_{test}.

Type	Precision	Recall
IP C&C	99%	99%
Domain C&C	100%	83%
Proliferation	100%	99%
DoS Attack	100%	99%

period of 1 second, and ii) more than 50 packets is sent from a "spoofed" source IP address (a source IP from the actual device), to multiple different destination IP in a period of 1 second. There is a slight difference between this type of traffic and the **proliferation** traffic; we distinguish the two according to the source IP address which will be the same as the device's in the proliferation phase, while in the "reflection" type of DoS traffic it will have a different source address.

Our heuristics classify communications with at least 99% precision and at least 83% recall. We evaluate the performance of our heuristics using our D_{test} =300 dataset and show the results in Table 4.1. Although the results indicate our heuristics are effective in identifying the type of communication, but there is a lower Recall in identifying the C&C server based on domain. This drop on Recall (83%) is due to use of DGA functionality to find the C&C server. Although our heuristics was able to identify the fixed domain name from the same malware, however it failed to identify the DGA generated domain since it does not maintain the connection with it.

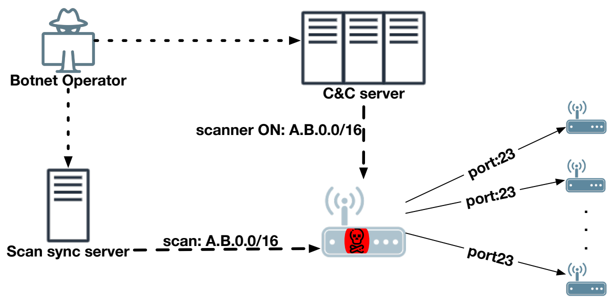


Figure 4.3: Host scanning of a subnet directed by the botnet operator to the botnet or to an external synchronizing server.

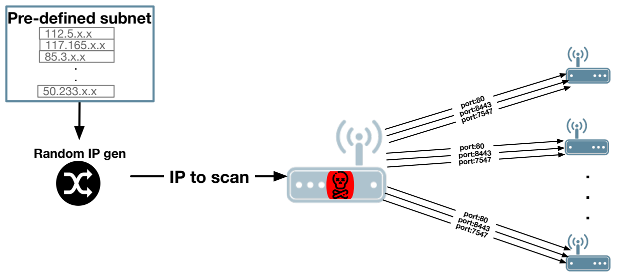


Figure 4.4: Host scanning randomly generated IP addresses from predefined subnets on one or multiple ports.

4.2 Study of IoT Botnets

We conduct extensive number of dynamic execution with network interaction. Specifically, we analyzed each of the 2885 malware binaries from our dataset and through multiple iterations we collect information about: i) malware proliferation techniques and how the exploits used, ii) different techniques used to connect to the server and/or to the botnet, and iii) communication protocol and the control commands that they have.

4.2.1 Malware proliferation

In our study we identified malware that perform propagation. We study this behavior based on two main characteristics: 1) Host scanning and 2) propagation mechanism.

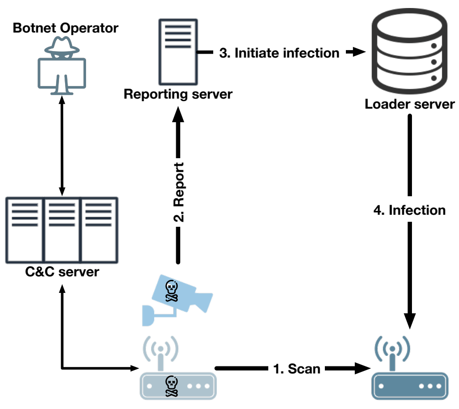


Figure 4.5: Proliferation mechanism as a built-in function in malware to enable rapid growth.

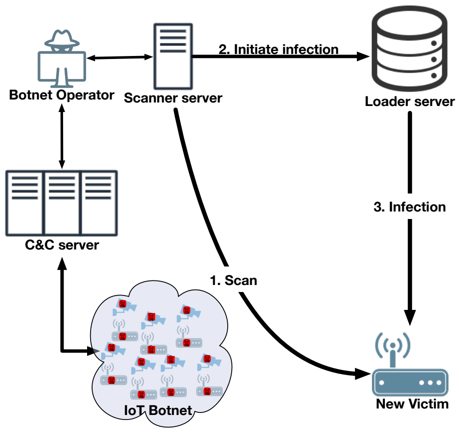


Figure 4.6: Proliferation mechanism as an external function controlled by the botnet operator to enable selective growth.

1. We identify 2 main host scanning techniques. In Figure 4.3 and Figure 4.4 we show the IP address selection for a potential device scanning varies:

a) In Figure 4.3 the botnet operator instructs the bots to scan a subnet to recruit potential vulnerable hosts. In some cases, the operator uses a separate "synchronizing" server which the botnet uses to coordinate their host scanning with the rest of the scanning.

b) In Figure 4.4 the malware scans a host on one or multiple ports by randomly selecting an IP address from a pre-defined list of subnets. This behavior can be attributed to the botnet that recruits more diverse set of devices from a specific network regions.

2. We identify 2 main propagation techniques. In Figure 4.5 and Figure 4.6 we show the main propagation techniques observed in our study:

a) Figure 4.5 shows the propagation technique that uses a "reporting server" to keep track of the infected devices and potentially vulnerable devices. Upon identifying a potentially device, the malware sends a payload that exploits the vulnerability which will inject an infection vector that will download a malware from a loader server and executes it. This propagation technique commonly used in Mirai IoT malware. In other malware families we did not observe a reporting server, however the "loader server" is an inseparable part of the propagation in IoT malware.

b) Figure 4.6 shows the malware propagation is done via an external "scanning" server instead of having the botnet to recruit the devices. As it appears such malware do not recruit a large number of devices hence a smaller botnet. This behavior has been observed in less known malware family such as TheMoon[191] and Moose [36].

In this study we identified *Loader server* that hosts malware binaries which is used as an important part of the IoT malware propagation. In most cases this is a separate server from the C&C server. Further investigation suggests that the malware developers use this server to host malware binaries compiled with different CPU architecture (such as MIPS and ARM) to enable infecting as much IoT devices as possible. This has been observed by Mirai botnet as well [20].

4.2.2 C&C server infrastructure

We investigated the different techniques observed in our IoT malware dataset in finding and connecting to the server and/or join the botnet. We show a breakdown of these techniques in the following.

1. We observe single C&C server infrastructure. The majority of the IoT malware binaries in our dataset use one technique to connect to their C&C server. About 2458 of the malware binaries use hardcoded IP addresses to connect with their server. In Some cases malware that use domain names to resort to a stronger C&C infrastructure

2. We observe C&C server via domain names. Aside from using IP addresses, there are malware that use domain names to resort to a stronger C&C infrastructure; botnet operator can change the IP address behind the domain name easier. Along with that we observe

Table 4.2: Malware approach for finding the C&C server.

IP address only	Single	2170
	Multiple	46
Domain only	Fixed	231
	DGA	2
Both		207

4.2.3 Malware communication protocol

As previously mentioned in before, there IoT malware source code available We observe 5 main botnet architectures that we show in Table 4.3.

4.2.4 Effectiveness of C&C impersonation

A key novelty of our work is that it can automate the engagement with a malware by impersonating its C&C server, as we explained in Chapter 2.

Key Result 1: We achieves 79% C&C impersonation success. We are successful in impersonating the C&C server for 79% of the binaries as shown in Table 4.5. Our approach makes the malware follow botnet commands, using the process which we described in Section 3.2.5.

We also study the number of binaries that responded to different types of commands in Table 4.6. 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.

Source Name	Proliferation			C&C Type.	Functions	
	Type	Host Scan	Infection		DoS	Other
Gafgyt	Built-in	Random IP	Brute-force	Agent, IRC	HOLD, JUNK, UDP, TCP	Kill malware, Execute shell, Stop attack
Kaiten	Built-in, Server	Subnet defined by C&C	Brute-force	IRC	HOLD, JUNK, UDP, PAN, UKNW, HTTP	Kill malware, Stop attack
Tsunami	Built-in	Random IP	Brute-force, Exploit	IRC	STD, UKNW, SNMP, TSUNAMI, NTP, PAN, BLACK, XMAS, PHATWONK	Execute shell, Update malware
Remaiten	Built-in	Subnet defined by C&C	Brute-force	IRC	STD, UDP, UKWN, TCP	Kill malware, Stop attack
Mirai	Built-in	Random IP	Brute-force	Agent	UDP, TCP, HTTP, GRE	Kill malware, Stop attack

Table 4.3: The 5 malware source code family identified by manually analyzing 174 source code.

Family from	Impersona-	Gafgyt C&C		Tsunami C&C		Aidra C&C	Mirai C&C
Virustotal	tion Success	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 4.4: 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.

Total binaries	2885	
Activated	2688	93%
Engaged	2291	79%

Table 4.5: We can impersonate the C&C server for **79%** of the binaries. Versions of the Gafgyt and Mirai of communication protocols are most widely used.

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

Table 4.6: 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 2: 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 earlier. In Table 4.4, 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. Intrigued by this, we found at least one hybrid binary among the 5 Gafgyt binaries mentioned above. The binary uses functions such as a telnet scanner, known to be used in Gafgyt family, while it uses the C&C communication protocol of the Mirai family. In the future, we will further investigate this interesting phenomenon.

Key Result 3: 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 [94].

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 [160]; they setup a proxy on an infected device, which can provide HTTP and SMTP services [12].

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 [6]. We also found malware that prohibits all network communication: it installs firewall rules to drop all incoming and outgoing packets.

4.2.5 Malware behavior in host

We highlight some interesting behaviors from our DMain malware dataset. Inspired by the *MITRE ATT&CK* framework [130], we show a break down of the common techniques used in each of the malware operation in Table 4.7 with the number of binaries exhibiting each behavior.

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

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 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 [35].

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

```
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:               # generic 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.

daemon.mipsel.mod or lightaidra.

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 out 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.

4.3 Related Work

We review related work in the space of analyzing IoT botnet traffic through the following broad categories.

a. Analyzing IoT malware: There are a few number of open source sandboxes focusing on executing and analyzing Linux malware binaries and their behaviors such as Limon [120], and Detux [65]. Padawan [57] recently released was released as an online malware analysis platform. The main purpose of these projects is to analyze the behavior of the malware on the OS level without profiling them. In this work we focus on the networking aspect of IoT malware and deliver longitudinal analysis on a great number of malware binaries.

b. Static analysis of malware: There has been an extensive effort in analyzing the structure and the behavior of a given malware binary. The majority of the effort in static analysis is focused on PC and smartphone malware [52, 124, 132, 163, 16]. However, static analysis for malware binaries has known limitations [133]. In this work we make use of dynamic analysis technique to impede the challenges in static analysis.

c. Dynamic analysis of malware: Besides static analysis technique, there are literature in dynamic analysis of malware, which focus on PC and Android malware [115, 109, 110, 106]. In this work our focus is to dynamically analyze the malware to understand its behavior in OS and networking level. In addition, our focus is on IoT malware which poses several challenges that requires novel techniques and tools.

d. malware network traffic analysis: Several effort has been established in understanding malware traffic to identify trends and growth of such malicious traffic [119, 73, 114, 118, 117]. In addition to that there has been several effort to model and profile botnet traffic [97, 190, 84, 85, 37, 54, 83, 66]. However, compared to our work these efforts make use of the live traffic of the malware. In this Chapter our focus is to analyzing the malware and engaging in communication to profile its behavior.

4.4 Conclusion

The goal of this chapter was to perform a longitudinal on the IoT malware binaries to profile its behavior in both OS and networking level. Our work sheds light into the unknowns of IoT malware behavior and undertake the challenge of hard to analyze IoT malware.

Our main contribution is that we perform in depth analysis on the behavior of IoT malware. We map the lifecycle of any given malware to the techniques and tactics introduced by MITRE ATT&CK framework [130]. We identified common techniques such as defense evasion and persistence process. In addition, we explore the networking behavior of such malware and identify the common techniques used to communicate with their C&C server. We engage with the malware as its C&C server and issue commands to have the malware engage in malicious activity in the contained environment. Moreover we identified cross-talk behavior between different families of malware indicating that IoT malware may have been classified incorrectly. One of the immediate results of our work is the TBs of attack traffic generated from our analysis.

Our work is a first one of its kind to tackle the unknown threats of IoT malware threat head on and discover common techniques used by such malware. We believe the outcome of this work can be used to propose defense mechanisms against such threat.

Chapter 5

Systematic Analysis of IoT Malware Behavior in Target Environment

It is evident that the malware targeting IoT devices has grown exponentially [72] with the emergence of Internet connected IoT devices and the widely available malware source code [129]. Streamlining the analysis of such threat has been proposed by researchers for a faster mitigation [61, 57]. The idea is to makes use of pre-built virtual machines to analyze streams of malware binaries to extract actionable information such as malware Indicator of Compromise (IoC). However, this approach fails to address the diversity of IoT devices which stands which stands as a key challenge in IoT and firmware analysis [204, 88]. Moreover, using pre-built VMs to emulate all possible IoT environment is not feasible [46] and does not scale [202].

The problem: How can we effectively analyze IoT malware with behavior sensitive to the target platform? This is the problem we tackle in this Chapter. We focus on performing dynamic execution to analyze the malware behavior. More specifically the goal is to: a) identify the techniques that an IoT malware uses to identify the type of the platform it is in, and b) measure the changes in the behavior of the malware being analyzed under different environment. Following we layout the challenges faced in solving this problem

Challenge 1. Identifying target environment requires analyzing the dynamic behavior of the malware and observe its interaction with the resources in the analysis environment. Throughout its life cycle, a malware binary invokes large number of API calls to the OS via system calls to access the different resources. Depending of the duration of the analysis the number of system calls may be in the order of hundreds of thousands to millions. The challenge is to pinpoint the system calls that malware may use to either verify or identify its target environment.

Challenge 2. Measuring the changes in the behavior is a crucial step in detecting and analyzing environment sensitive malware. As previously mentioned in the above, the number of system calls in each execution exceeds hundreds of thousands. On the one hand detecting changes in behavior requires a reference execution trace [104] which is not defined or available for IoT malware. On the other hand identifying the changes in the invoked system calls is not a trivial task given that the execution trace may differ in different iteration of analysis under the same analysis environment.

At the time of writing this work, there has been limited efforts to address the issue of IoT malware behavior and its variability with respect to target environment. We split the most relevant work into the following categories: (i) studies concerning malware sensitive to analysis environment [27, 104, 121, 116], (ii) force executing malware functions [199, 132, 152, 110, 198], (iii) analyzing environment evasion techniques [106, 107, 77, 147], and (iv) studies on malware behavior analysis [28, 53, 158, 159, 28].

In this chapter, we develop a tool to efficiently analyze IoT malware behavior in different target platform. To achieve this we record the behavior of the malware when interacting with the analysis environment. This includes any data exchange between the malware process and the OS kernel which is observed via system calls. we profile all the resource requests that malware performs which includes device drivers, files, network access, and other similar interactions. Next, the malware will be executed in a series of sandboxes to accommodate the malware's resource request. Later we compare the malware behavior in each execution to profile the changes in their behavior.

We show an overview of the tool architecture in Figure 5.1. Our tool uses RIoT-MAN (Chapter 3) as the means to execute the malware in different analysis environment and collect the execution traces.

The contributions of this work is as follows:

- We propose a novel approach to identify IoT malware sensitivity to the target environment. We record malware behavior based on its interaction with the OS kernel and detect any requests to the target environment including accessing files and drivers.

- We profile the changes of behavior of the IoT malware under different target environment. We propose a technique to first analyze the malware behavior under a reference

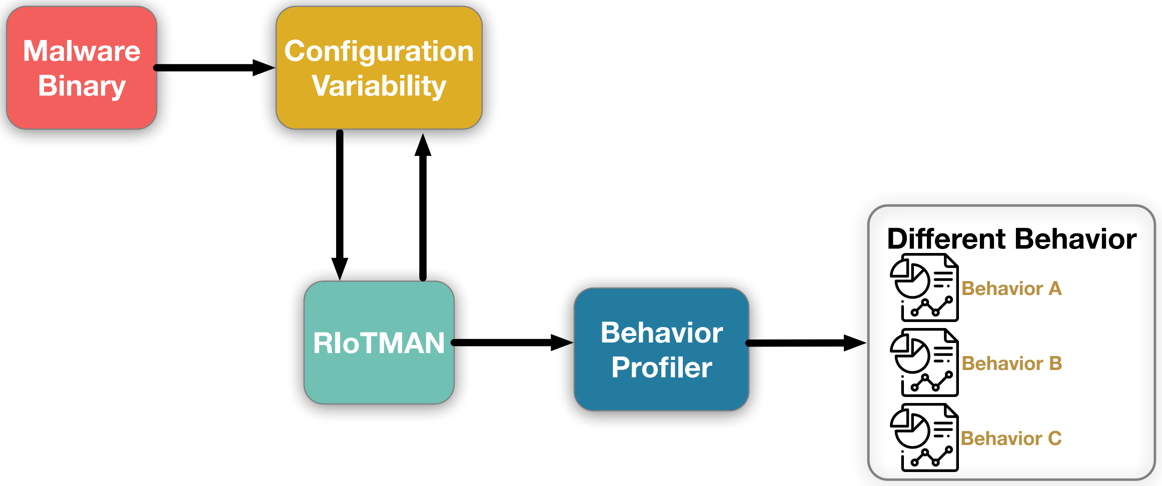


Figure 5.1: The overview of the tool. In this tool we are using RIoTMAN from Chapter 3.

sandbox and second identify changes in its behavior when executed under different simulated target environment.

- We demonstrate that our approach is able to identify malware that exhibit 2 to 8 different behaviors under different target environments. We identify malware that use "query and infect" and malware that terminate execution if identified analysis environment artifacts.

5.1 System Design and Implementation

We used **RIoTMAN**, our analysis tool in Chapter 3, and added two modules: 1) Configuration Variability, and 2) Behavior Profiler. The two modules work together to create different configuration setup and profile changes in malware execution. Below, we explore the technical details of our approach in this work.

5.1.1 Configuration Variability module

The purpose of this module is to direct RIoTMAN for creating different emulation setup for analyzing a given malware. This is based on the resource requests the malware makes on the infected device. By iteratively making each resources available, RIoTMAN creates a new analysis environment for the malware. In this work we focus on resources that the OS provides to the malware which includes filesystem and network communication by observing system calls invoked by the malware. Following we explain each of these resources and how we collect the information about each interaction.

A. Monitoring resource requests

The goal is to monitor any resource requests to the target environment that a malware makes in order to identify the resources that may change the behavior of the malware. In our desig, we use RIoTMAN to execute the IoT malware and record its execution traces which includes system call traces and networking traffic. We analyze malware behavior in the target environment to detect all the resource requests it makes. Our assumption is that we can monitor this via the malware's API call to the OS kernel. Although there are malware that may make use of kernel module and eliminate the need for API calls to the OS, however, to this date no IoT malware was observed to install a kernel module to conceal its malicious activities [57]. In our work, we focus on the following group of system calls shown in Table 5.1:

Filesystem: Malware may use filesystem resources to establish foothold or make itself persistent [130]. At the same time, it can use this resource to verify the target environment. For example, in different distribution of Linux OS the rc file may reside in different folder under different names which an adversary may use to identify the type of the infected environment.

Process: Malware may spawn multiple child processes to enable concurrency and avoid interruption in the main malware process in case any of the child processes stops due to any errors. The number and the purpose of each child processes work as an indication of how the malware may use the OS resources. For example, when a malware spawns multiple child processes to carry out the same behavior, it is an indication that whether that source is available or not does not intervene with the main execution.

Interactive Shell: This group refers to the system call that enable executing commands using the interactive shell in Linux such as `execve` system call [1]. In this work we consider this group of system calls as a separate category since malware developers may use this system call to use native shell commands on a given IoT firmware to carryout malicious functionality.

Networking: Malware uses network communication to receive control command from its server. Our goal is to identify the type of communication socket. For example, in a Mirai instance all communications are carried out via `raw` socket setup which is an indication that the adversary may not use known communication protocols, such as `TCP` or `UDP`.

[1] In their nature, such group of system calls are considered the same as **Process**, however, as opposed to `fork` or `clone`, `execve` does not copy of the main process and instead creates a new process.

Category	**Description**	**System calls**
Filesystem	Any actions on file system including read, write, create, and permission change	open() close() read() write() link() chown() create() mkdir()
Process	This includes creating processes, killing, cloning, and other similar actions	clone() fork() vfork() exit() getpid()
Interactive Shell	Any shell commands that are executed using interactive shell.	execve() execveat()
Networking	All of the networking system call which includes creating a file descriptor (`fd`) using socket to performing any data send and receive	socket() send() connect() socketopt() bind() listen() accept() recv() recvfrom() sendto()

Table 5.1: Group of system calls used to detect resource requests made by the malware.

B. Syscall Data Dependency Graph

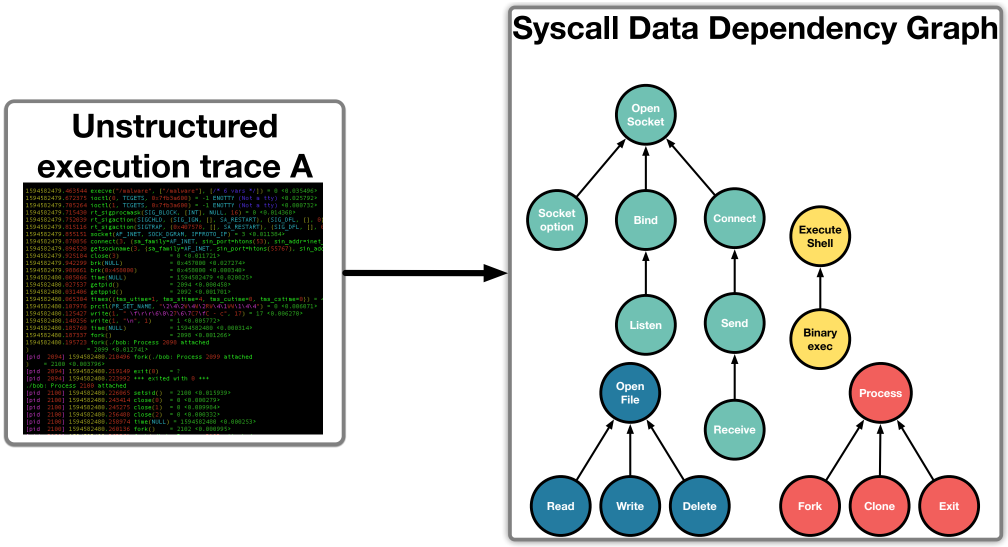
Automatically identifying the type and the name of the resource that a malware requests is a challenging task. As previously mentioned, depending on the analysis duration, each iteration of execution results in thousands of individual system calls. Each resource may be accessed differently, with a read or write type of command. Understanding such interactions with a given resource can not be retrieved in a trivial way from traces of the system calls. That is because when a userspace process accesses a particular resource, many system calls are invoked and the challenge is to identify the kind of access done on the resource by analyzing the many system calls. To tackle this challenge, we use Syscall Data Dependency Graph technique [109] to create graphs that represent resources and how they are being used.

In Figure 5.2, we show the conversion of the system call traces to multiple subgraphs that represent each resource and access to them. The root node is the resource and the other nodes are the actions on that resources. The connection between each node can be reconstructed using the arguments from each system call. All subgraphs are categorized into 4 groups which is shown in Figure 5.2: 1) filesystem subgraphs, 2) process subgraphs, 3) interactive shell subgraphs, and 4) networking subgraphs.

Following we demonstrate an example of how this method works. As shown in the following, this example consists of a process opening `fileA` to read from and write in it:

Listing 5.1: Example of opening file with readonly permission

```
open("fileA", READ_ONLY) = 2
```



```
1594582479.463544 execve("/malware", ["/malware"], [/* 6 vars */]) = 0 <0.035496>
1594582479.672375 ioctl(0, TCGETS, 0x7fb3a600) = -1 ENOTTY (Not a tty) <0.025792>
1594582479.705264 ioctl(1, TCGETS, 0x7fb3a600) = -1 ENOTTY (Not a tty) <0.000732>
1594582479.715430 rt_sigprocmask(SIG_BLOCK, [INT], NULL, 16) = 0 <0.014368>
1594582479.752039 rt_sigaction(SIGCHLD, {SIG_IGN, [], SA_RESTART}, {SIG_DFL, [], 0
1594582479.815116 rt_sigaction(SIGTRAP, {0x407578, [], SA_RESTART}, {SIG_DFL, [],
1594582479.855151 socket(AF_INET, SOCK_DGRAM, IPPROTO_IP) = 3 <0.011384>
1594582479.870856 connect(3, {sa_family=AF_INET, sin_port=htons(53), sin_addr=inet_
1594582479.896520 getsockname(3, {sa_family=AF_INET, sin_port=htons(55767), sin_add
1594582479.925184 close(3)                = 0 <0.011721>
1594582479.942299 brk(NULL)               = 0x457000 <0.027274>
1594582479.988661 brk(0x458000)           = 0x458000 <0.000340>
1594582480.005066 time(NULL)              = 1594582479 <0.020825>
1594582480.027537 getpid()                = 2094 <0.000458>
1594582480.031406 getppid()               = 2092 <0.001701>
1594582480.065304 times({tms_utime=1, tms_stime=4, tms_cutime=0, tms_cstime=0}) = 
1594582480.107976 prctl(PR_SET_NAME, "\2\4\2W\4W\2RW\4\1WW\1\4\4") = 0 <0.006071>
1594582480.125427 write(1, " \f\r\r\6\0\27\6\7C7\fC - c", 17) = 17 <0.006270>
1594582480.140256 write(1, "\n", 1)     = 1 <0.005772>
1594582480.185760 time(NULL)              = 1594582480 <0.000314>
1594582480.187337 fork()                  = 2098 <0.001266>
1594582480.195723 fork(./bob: Process 2098 attached
)                 = 2099 <0.012741>
[pid  2094] 1594582480.210496 fork(./bob: Process 2099 attached
)    = 2100 <0.003796>
[pid  2094] 1594582480.219149 exit(0)    = ?
[pid  2094] 1594582480.223992 +++ exited with 0 +++
./bob: Process 2100 attached
[pid  2100] 1594582480.226065 setsid()  = 2100 <0.015939>
[pid  2100] 1594582480.243414 close(0)  = 0 <0.000279>
[pid  2100] 1594582480.245275 close(1)  = 0 <0.009984>
[pid  2100] 1594582480.256488 close(2)  = 0 <0.000332>
[pid  2100] 1594582480.258974 time(NULL) = 1594582480 <0.000253>
[pid  2100] 1594582480.260136 fork()    = 2102 <0.000995>
```

Figure 5.2: Conversion of traces of system call to subgraphs representing the interaction with each resources. Resources are the root of each graphs.

```
read(2, "fileA␣content", 13) = 13

write(2, "write␣fails", 11) = -1
```

The output from `open` system call is the `fd` id (which is 2 in this example) that will be used as the first input argument for both `read` and `write`. It is important to note that different arguments may be inputs or outputs of a system call. In this example, the second argument for `read` and `write` are output result and input argument respectively. In Figure 5.3 we show how this trace is converted to the resource graph.

5.1.2 Behavior Profiler Module

One of the key novelties of this work is the the Behavior Profiler module. The purpose of this module is to profile the execution trace from each iteration of analysis and utlimately compare different execution of the same malware to identify the changes

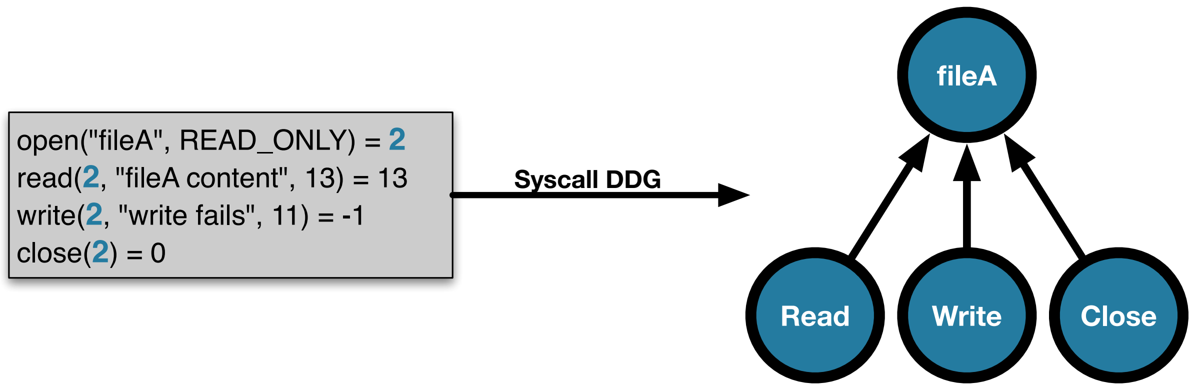

```
open("fileA", READ_ONLY) = 2
read(2, "fileA content", 13) = 13
write(2, "write fails", 11) = -1
close(2) = 0
```

Figure 5.3: The example of accessing `fileA` syscall trace to data dependency graph.

in malware behavior. In doing so we use the system call data dependency graph to both profile the behavior and compare the execution traces.

An overview of the steps in this module is depicted in Figure 5.6. The input is the execution traces collected from different iteration of analysis in RIoTMAN and convert them to structured subgraphs of interactions with system resources which will be used to identify the differences between each execution.

Below, we explain the steps in profiling the behavior to use in identifying the changes in malware behavior:

Remove Noise: A malware may invoke system calls that may differ across different iteration of analysis under the same environment. These includes listing directories, listing running processes, and similar. The goal is to remove such traces when comparing the behavior of the malware across different iteration of analysis. As we will show later in our evaluation, this step helps with reducing the similarities between

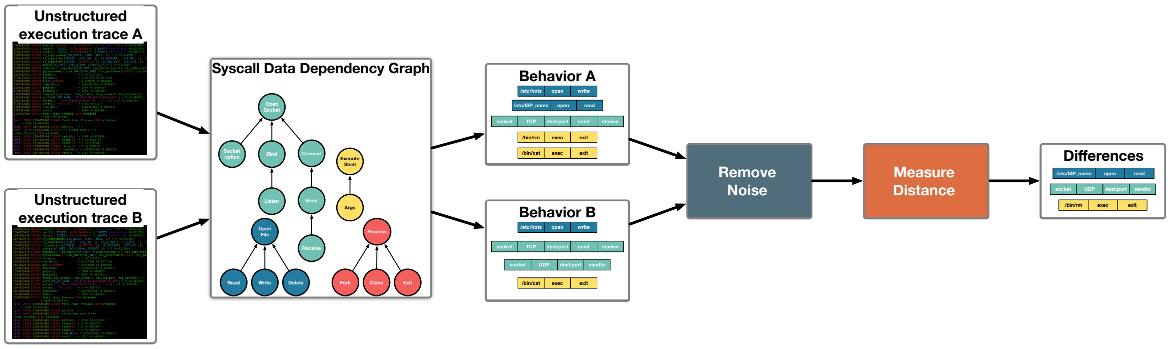


Figure 5.4: The overview of the Behavior Profiler module.

executions that exhibit different behaviors.

Flatten Graphs: Comparing graphs is not a trivial task as it is known to be an hard problem. Given the nature of this work focuses on identifying the differences in resource usage when comparing malware behavior under different target environment, we resort to flattening the subgraphs to create strings that represent actions on each resources. The root node represents the name of the resource and we traverse the rest of the nodes (which indicate an action on the resource) based on their timestamp. Note that each subgraph may belong to one of the 4 categories of filesystem, process, interactive shell, and networking. If the a resource with the same sequence of actions is being accessed multiple time in an iteration of execution, an additional number will be added to the profile to indicate the repetition of that particular behavior. The output of this step will be referred to as the profile of a given execution trace.

Remove Similarities: For calculating the similarities between each iteration of execution, in our algorithm we remove the similarities in behavior. A similarity in behavior refers to a series of action on a given resource (subgraphs) that appear with the exact same sequence of accesses and similar actions. We identify these by exact matching each flatten subgraphs. As we show later, this helps with identifying changes in malware behavior executed under different target environment.

Measure Sequential Similarities: The remaining subgraphs after removing the similar behavior, represent the differences in malware behavior analyzed under different target environment. In this work we use Ratcliff et. al.[156] to measure the similarity score between each flatten graphs. More specifically, given two sequence of the flatten graphs, a similarity score is consists of the number of matching nodes determined by longest common substring (LCS) in addition to recursively performing LCS on the matching and non matching nodes. Therefore, identical behavior will have a score of 1 (or 100%) and the less identical the lower the score will become. We set a threshold for the similarity score, determined empirically, that below which indicates variation in the given two behavior. We resort to this matching algorithm since it is robust against slight difference between matching nodes. For example, in case of networking behavior a source port number for a communication may be different in each execution and the goal is to avoid such variation be tolerated when measuring the similarities. In addition to the similarity score, this algorithm provides the list of insertion and deletion of nodes to help identify the changes in the behavior of malware under different analysis environment.

As shown in Figure 5.5, the outcome of these step is a set of unmatched behavior that had a similarity score of less than the threshold.

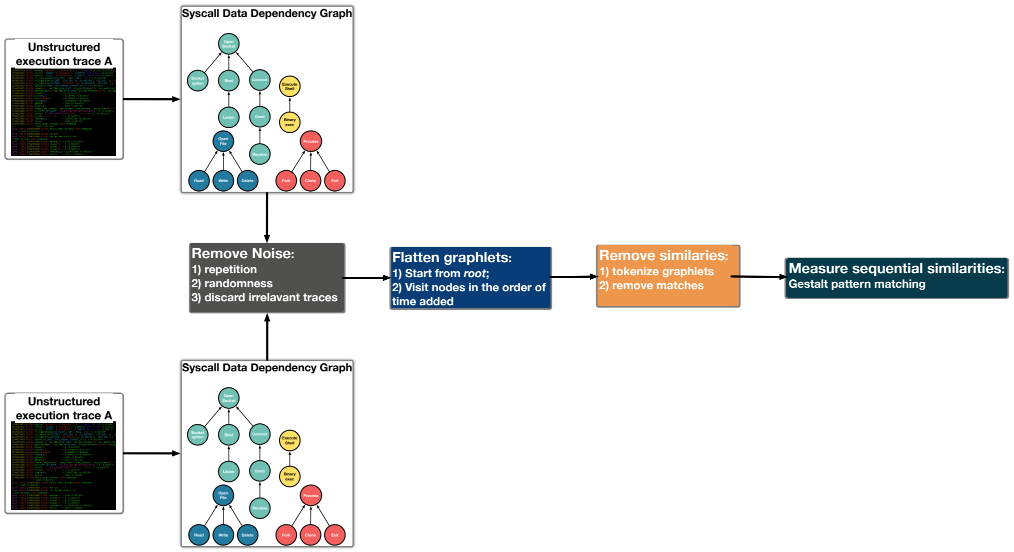

Unstructured execution trace A
Syscall Data Dependency Graph
Remove Noise:
1) repetition
2) randomness
3) discard irrelavant traces
Flatten graphlets:
1) Start from *root;*
2) Visit nodes in the order of time added
Remove similaries:
1) tokenize graphlets
2) remove matches
Measure sequential similarities:
Gestalt pattern matching
Syscall Data Dependency Graph
Unstructured execution trace A

Figure 5.5: The steps taken to compare the execution traces of a given malware under two different analysis environment.

Similarity Threshold

We establish the similarity threshold when comparing the behavior of malware empirically. We randomly selected 2000 malware binaries and executed each 4 times under the same analysis environment as the one determined by RIoTMAN. In Figure 5.6 we show the frequency of similarity score of each execution compared to itself. As it appears, the distribution of similarity score is skewed towards complete similarity (100%) in behavior. We notice that the lowest similarity score is close to 90% for a small number of malware. Our manual investigation determined this variation in malware behavior is due to randomly generated `nick` for the IRC channel as well as source port number used in communication. Such small differences in the behavior similarity score does not lead to changes in malware behavior. Given this chart and the minimum similarity score being close to 90%, we selected that as the threshold for determining changes in malware behavior.

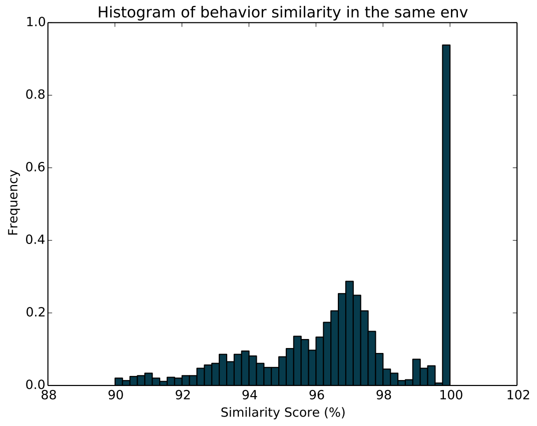


Figure 5.6: Similarity score calculated for 2000 malware that are executed 4 times under the same environment determined by RIoTMAN. The similarity score varies between 90% to 100%.

5.2 Evaluation

In this section we evaluate our tool using 2885 real world IoT malware binaries from Chapter 4. The goal is to identify variations in malware behavior when the analysis environment is configured differently. More specifically our goal is to answer the following questions:

- What techniques do IoT malware use to verify the target environment?
- How do changes in analysis environments result in variations in malware actions?
- What variation in malware actions constitute to changes in malware behavior?

We explore each of the questions in details to verify the effectiveness of our analysis method.

Verifying target environment: The first problem in analyzing IoT malware sensitivity to target environment is to identify how they confirm the environment they are in. As explained earlier in Section 5.1 to tackle this challenge we focus on the interaction with the following resources on an analysis environment: filesystem, process, interactive shell, and networking. We use the generated subgraphs of malware interaction with the environment resources to identify its technique in verifying an environment. These resources can not be any of the following: 1) their absence may result in failing to activate the malware (see Chapter 3), and 2) such resources can not be the default resources files available in all Linux distribution (e.g. /etc/hosts).

Changing target environment: Given a list of resources that malware has interacted with in its execution trace, the goal is to make each resources available in a series of analysis experiments. The goal is to have the malware be able to access each resources per each execution which leads to a new analysis environment i.e. target environment.

Identifying changes in behavior: Using the similarity algorithm defined in Section 5.1, we use the unmatched nodes as the grounds to claim changes in malware behavior. As we show later in Case Study, upon the availability of each resources per execution, the malware exhibits additional behavior which is an indication of change in behavior. These changes are in fact series of system calls that are observed only in the appropriate analysis environment.

Resource requests

We found **981** unique resources requested that were made by malware binaries since they are not available in the *generic* platform. In Table 5.2 we categorize the resource requests based on their type and how they were resolved.

Resource Request		From database	"Dummy" file
Total		623	358
Software Resource	**Lib**	171	-
	Shell	189	40
	LKM	10	1
	Config	245	317
Hardware Resource	**NVRAM**	5	-
	Other	3	-

Table 5.2: Summary of resource requests made by the our dataset of IoT malware. If the resource is not available in *Configuration database*, RIoTMAN creates a "dummy" file appropriately constructed per request type.

I. **Software Resource**: There have been 973 of such requests. 615 of such requests were resolved through the use of *Configuration database* (Chapter 3) which includes library files specific to IoT devices e.g. `libcms_msg.so` and `libnat.so` or configuration files like `ISP_name`. On the other hand in 358 cases the resource was resolved with "dummy" files (details in Section 3.2): (1) files related to scanning; e.g. `login` file that holds list of credentials for bruteforce scanning, (2) specific configuration file related to the malware; e.g. `Zopee3ve` is requested by an instance based on `linux.wifatch` [179], or (3) in one case malware loads (using `modprobe`) `hi_rtc.ko` kernel module specific to *HiSilicon*[2] IP camera.

II. **Hardware Resource**: We identified 8 types of such request. Interestingly, we observed malware binaries that tampers with the `watchdog` timer to prevent the device from rebooting. In other cases the malware requests to modify or read configuration from

[2]http://www.hisilicon.com/

the NVRAM directly or through the use of shell commands such as `nvram` or `cfgmtd`.

5.2.1 Case Study

In this section, we explore 3 case studies of: a) query and infect, b) analysis environment evasive, and c) hidden behavior. We show that IoT malware behavior differs depending on the target environment since there is a plethora of diverse sets of IoT devices which serve different and unique purposes.

Query and infect

In this category, the malware infects an environment by first *query*'ing the configuration setup. Originally, this technique was observed in APT malware targeting PC [199]. In this study, we observe similar behavior in 11 of the IoT malware in our dataset which exhibit 8 distinct behavior related to persistent process. Below, we detail this behavior using one of the malware binaries in this category.

In this example, the malware queries series of configuration file resource which was observed in its behavior subgraphs. None of the resources are available in a basic Linux distribution. With further investigation we are able to associate each configuration file to a group of commodity devices as we show in Table 5.3.

The Configuration Variability module is responsible to inform RIoTMAN to make each resources available per iteration of execution. Figure 5.7 depicts each iteration of the analysis execution under different target environment configuration. Table 5.4 compares the number of system calls invoked per each execution under 8 distinct configuration setup

Configuration file name	**Commodity device with the file**
`/etc/init.d/rcS`	Older Linux devices, raspberry Pi
`/usr/sbin/nvram`	Home routers such as Linksys
`/bin/cfgmtd`	Ubiquity Network devices
`/mnt/Config`	ZyXEL home routers
`/sbin/sncfg`	D-Link routers
`/etc/ISP_name`	Seowon Intech WiMAX
`/etc/Model_name`	Unknown home routers

Table 5.3: List of configuration file resources and the name of the commodity devices that host them.

from None, indicating none of the configuration files are available, to Conf-0 to Conf-7 and All which hosts all the configuration files.

Using the similarity measure introduced in Section 5.1, we compare the malware behavior across all the iteration of execution. In Figure 5.8, we show a heatmap that shows the similarity score between executions compared pairwise. The lower the similarity score the less similar behavior they exhibit.

We further investigate the mismatch between the behavior from each iteration and identify that this group of malware takes advantage of commodity device **Interactive shell** to make itself persistent. We investigate the shell commands that were executed in each iteration of analysis and observe 69 commands with exact same parameters have been observed across all of the iteration of analysis. Additional commands are executed

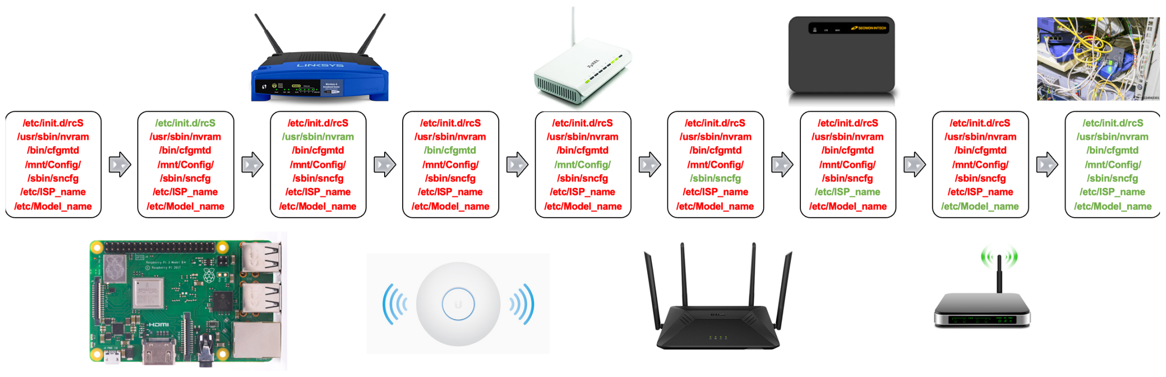
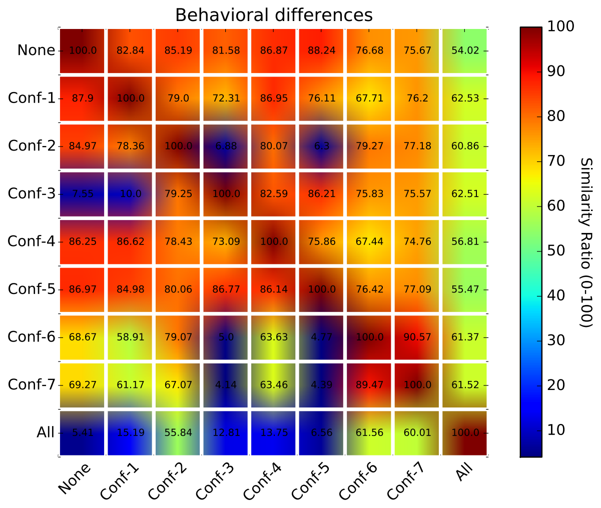


Figure 5.7: Through 8 iteration of execution in this example, the Configuration Variability module makes one of the configuration resources available (colored in green) and exclude other (colored in red) in each iteration of analysis.



	None	Conf-1	Conf-2	Conf-3	Conf-4	Conf-5	Conf-6	Conf-7	All
None	100.0	82.84	85.19	81.58	86.87	88.24	76.68	75.67	54.02
Conf-1	87.9	100.0	79.0	72.31	86.95	76.11	67.71	76.2	62.53
Conf-2	84.97	78.36	100.0	6.88	80.07	6.3	79.27	77.18	60.86
Conf-3	7.55	10.0	79.25	100.0	82.59	86.21	75.83	75.57	62.51
Conf-4	86.25	86.62	78.43	73.09	100.0	75.86	67.44	74.76	56.81
Conf-5	86.97	84.98	80.06	86.77	86.14	100.0	76.42	77.09	55.47
Conf-6	68.67	58.91	79.07	5.0	63.63	4.77	100.0	90.57	61.37
Conf-7	69.27	61.17	67.07	4.14	63.46	4.39	89.47	100.0	61.52
All	5.41	15.19	55.84	12.81	13.75	6.56	61.56	60.01	100.0

Figure 5.8: Heatmap of similarity for the same malware under 8 different configuration. The score ranges from 0 to 100% indicating least to total similarity in malware behavior.

	None	Conf-1	Conf-2	Conf-3	Conf-4	Conf-5	Conf-6	Conf-7	All
Exec time	1333	1353 s	1299 s	1341 s	1378 s	1332 s	1347 s	1368 s	1376 s
total syscall	17139	17590	17428	17756	17809	17417	17987	18050	20222
# of Process	74	82	82	85	82	81	89	89	136
# of Execve	71	79	83	81	79	79	86	86	136
# of shell cmd	69	77	78	77	77	77	82	82	124
# of Sockets	133	135	129	134	137	133	134	136	137
# of Connect	1304	1321	1264	1308	1345	1298	1313	1333	1337
# of Open	27939	34232	33324	35834	42192	33053	61065	48119	67527
# of file access	25	26	25	25	27	26	28	28	32
# fail Open	14	13	14	13	14	13	14	14	9
# success Open	11	13	12	14	14	16	16	14	24

Table 5.4: The number of system calls changes as the configuration varies. The same malware is exhibiting different behavior.

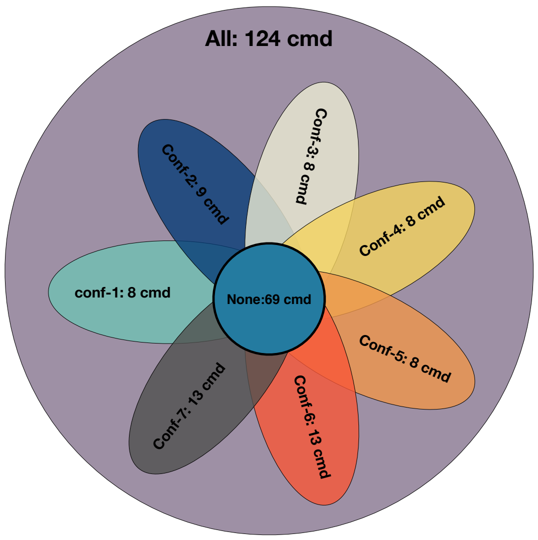


Figure 5.9: Venn diagram of intersection of shell commands across each analysis iteration. 69 commands are commonly observed in all iterations and a total 124 commands was observed in a system with all configuration.

depending on the availability of particular resources. Interestingly, if all resources are available (in analysis iteration All) all of the shell commands are from all the other iterations is observed. We show a diagram of shell commands observed in each iteration of analysis in Figure 5.9.

Observation: In this category of malware, we observe unique and distinct behavior depending on the target environment configuration, which we highlight here:

- **Conf-1**: malware makes itself persistent as part of the firewall rule setup.
- **Conf-2**: malware downloads a copy of itself after 2 minutes.
- **Conf-3**: malware makes itself persistent and after 3 months reboots the device.
- **Conf-4**: malware creates an additional function to the timezone synchronization function of the device to download a new version of itself.

- **Conf-5**: malware makes itself persistent by adding a custom startup script to its NVRAM.
- **Conf-6**: the persistent process is added as part of a new file under the `/etc/init.d/S99nvrak`. It uses the file content from `/etc/ISP_name` to create a nickname to connect with C&C IRC server.
- **Conf-7**: Same as above, however, it uses `/etc/Model_name` to generate a nickname.

Besides the change in malware behavior observed via OS behavior, there are change in the communication as well. Our analysis have determined that depending on the type of target environment, the malware may take longer time to initiate the first communication packet to its C&C server. Further investigation shows that the reason in delay is the longer persistent process with respect to the target environment. We show a break down of the seconds taken from starting of the malware execution until the first C&C packet is initiated in Table 5.5.

	None	Conf-1	Conf-2	Conf-3	Conf-4	Conf-5	Conf-6	Conf-7	All
First comm packet	10 s	20 s	18 s	18 s	19 s	18 s	20 s	20 s	23 s

Table 5.5: The malware may take longer to initiate its first communication packet. The more general the infected target environment is the less time it takes for the malware to start engage in communication.

Hidden behavior

This category of malware describes a greater group of malware that have "hiddern" functionalities which will reveal depending on the target environment. We define such IoT

	None	Conf-1
# of Process	14	15
# of Shell cmd	13	13
# of Socket	11054	284
# of Connect	1091	1494
# of Open	49	366
# of File access	18	35
# of Success file open	10	27

Table 5.6: In this example, the malware behavior changes in the number of sockets created when different configuration setup is available for it.

malware as malware with *hidden behavior*, which refers to a class of behaviors that will emerge only if the target environment meets certain criteria. We highlight one of such malware below.

In this example, the malware requests to access resources that are not available in a basic Linux environment native to MIPS architecture. It requests to access NS lookup library files under `/lib/mips-linux-gnu` directory which indicates a non native MIPS environment. Upon the availability of the library files under the certain MIPS architecture, the malware reveals a new behavior which is to start initiating connection with a particular hostname. If the resource is not available, it resort to using a harcoded IP address to communicate with its server. Table 5.6 shows changes in the number of system calls invoked indicating changes in behavior.

Observation: In this category of malware, we observe hidden behaviors in different configuration setups that would have not been observed otherwise. In **Conf-1** the requested artifacts indicate a non native MIPS Linux environment. Our further investigation suggests that this is a technique employed by the malware to verify, in a naive way, that it is being executed via `qemu-mips-static` and `chroot`. This is a common practice by malware researchers when analyzing malware targeting non x86 CPU architecture. In this study, we are able to analyze the malware to reveal a behavior that it has for being executed under non native environment.

5.3 Related Work

In this section we explore the related work in the space of analyzing malware sensitive to target environment.

a. Analyzing environment sensitive malware: This group of malware analysis is the most relevant body of work to this chapter. So far, most studies have focused in analyzing PC or smart phone specific malware. Balzarotti et al. [27] focus on PC malware that are sensitive to analysis environment. In another study researchers have compared the execution traces of a malware in a reference environment compared to an analysis setup [104]. More studies have focused on changes in the behavior of PC malware with respect to target environment[121, 116]. In this work we focus on the IoT malware and having a plethora of target devices makes the problem challenging and unique compared to the previous studies.

Force executing malware functions: This category of work focuses on identifying and isolating malware functions to extract malicious function gadgets [110] or in some work [198] use malware communication function to identify their C&C server in the wild. Other works focuses on exploring different execution path of a given malware [199, 132]. Firstly the mentioned research work focus on PC malware and do not address IoT malware. Second, one of the key novelty of our work is to focus on resource usage of a given malware which makes the problem space limited to target environment and does not require exploring its execution path.

Analysis evasive techniques: To combat automated malware analysis, a body of work has been dedicated to identify analysis evasive malware [106] and analyzing evasive malware [107]. In addition to that a body of work is dedicated to understand the evasive techniques that malware can employ [77, 147]. In this work, focusing on IoT malware, we have identified malware with hidden behavior that may not have been invoked if their evasive technique was not recognized.

Malware behavior analysis: There is a tremendous amount of work dedicated in this space with mostly focusing on PC and smart phone malware[28, 53, 158, 159, 28]. The main focus on these study is to compare the malicious behavior of a malware to that of a benign one. In this work our main focus is to understand the changes in malware behavior when compared to a reference analysis execution. We focus on the interaction of a given malware with resources of the analysis environment and proposed techniques to identify the changes in their behavior.

5.4 Conclusion

The focus of this work is analyzing IoT malware behavior sensitive to target environment. We have propose and implemented novel and unique analysis techniques with the main focus on IoT malware. We tackle the difficult challenge of identifying environment sensitive malware and analyzing them.

The main contribution of this work is two fold: (a) we identify the techniques that IoT malware uses to identify target platform, and (b) we analyze such malware and identify changes in its behavior. we first analyze the malware using the reference platform, RIoT-MAN (Chapter 3, to identify the resources that a given IoT malware accesses throughout its execution. Second, we iteratively create different analysis environment using the information from the reference platform. Lastly we compare each of the iteration of execution to identify changes in the behavior of the malware. We observe malware with 8 different infection processes that depending on the type of target environment they exhibit unique behavior. Our approach is able to identify malware with hidden behavior that is revealed when the infected environment meets the target platform criteria.

With the results achieved in this study, we expect to observe more IoT malware sensitive to the target platform. We view our approach an important addition in battling IoT malware in the near future.

Chapter 6

Conclusions

The key novelty of this dissertation is that it: (a) focuses on IoT malware, and (b) provides unique capabilities to engage and communicate with the malware. We tackle each key challenges in IoT malware analysis which includes identifying malware target platform and engage in communication with it.

The contribution of this dissertation lies in the following: (a) an iterative adaptation approach to identify the target platform for a given IoT malware, (b) enable automation in analysis by using information from other IoT malware and the use of *Configuration database*, (c) identify environment sensitive malware and detect the changes in IoT malware behavior under different target environment. We combine in-depth analysis of dynamic behavior of the malware and its interaction with the target environment to explore IoT malware behavior and its characteristics.

The tools we implemented are successful at activating given unknown IoT malware binaries, engaging in network communication, and profiling their behavior. We successfully activate 93% of our malware binaries (2688 out of 2885 binaries) by dynamically create suitable analysis environment. We observe more than 900 different configuration setup is required to activate 2688 of all the malware binaries. In addition we are able to activate and observe malicious behavior from 173 malware binaries that are deemed as benign by the detection engines available in Virustotal reporting. We manage to impersonate the C&C server for **79%** of our binaries: in a contained environment we make the malware begin DDoS attacks, enter its proliferation mode, and trigger functions other. We observe "cross-talk" behavior in the malware family; a given malware maybe labeled as a particular malware family, however, its behavior is similar to another family. We enumerate the propagation and C&C server discovery techniques observed in our malware dataset.

With the rapid growth of Internet connected IoT devices, we expect to see more threats concerning such devices. We view our work as an important step towards analyzing threats towards IoT devices effectively and efficiently with the focus on IoT malware.

Bibliography

[1] Aisi malware statistics. Accessed: 2017-06-27.

[2] Anubis. *http://anubis.seclab.tuwien.ac.at/*.

[3] Soho wireless router (in)security. Accessed: 2017-09-22.

[4] Systemtap. *https://sourceware.org/systemtap/*.

[5] Threat advisory: Kaiten/std router ddos malware — akamai. Accessed: 2017-09-22.

[6] Permanent denial-of-service attack: Brickerbot. *https://ics-cert.us-cert.gov/alerts/ICS-ALERT-17-102-01A*, 2017-04.

[7] Openwrt linux distribution for embedded devices — soho routers. *https://openwrt.org/*, 2019-02.

[8] Iot inspector. *https://iotinspector.org/*, 2020-06.

[9] Padawan. *https://padawan.s3.eurecom.fr*, 2020-06.

[10] Tigist Abera, N Asokan, Lucas Davi, Jan-Erik Ekberg, Thomas Nyman, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. C-flat: control-flow attestation for embedded systems software. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 743–754. ACM, 2016.

[11] Tankut Akgul, Vincent J Mooney III, and Santosh Pande. A fast assembly level reverse execution method via dynamic slicing. In *Proceedings of the 26th International Conference on Software Engineering*, pages 522–531. IEEE Computer Society, 2004.

[12] Genshen Ye Alex.Turing. An Analysis of Godlua Backdoor, 2019. `https://blog.netlab.360.com/an-analysis-of-godlua-backdoor-en/`.

[13] Genshen Ye Alex.Turing. An Analysis of Linux.Ngioweb Botnet, 2019. `https://blog.netlab.360.com/an-analysis-of-linux-ngioweb-botnet-en/`.

[14] Hui Wang Alex.Turing. Mozi, Another Botnet Using DHT, 2019. `https://blog.netlab.360.com/mozi-another-botnet-using-dht/`.

[15] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1362–1380. IEEE, 2019.

[16] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. Graph-based malware detection using dynamic analysis. *Journal in computer Virology*, 7(4):247–258, 2011.

[17] Dennis Andriesse, Xi Chen, Victor Van Der Veen, Asia Slowinska, and Herbert Bos. An in-depth analysis of disassembly on full-scale x86/x64 binaries. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 583–600, 2016.

[18] Kishore Angrishi. Turning internet of things (iot) into internet of vulnerabilities (iov): Iot botnets. *arXiv preprint arXiv:1702.03681*, 2017.

[19] Anna-senpai. [free] world's largest net:mirai botnet, client, echo loader, cnc source code release. *https://hackforums.net/showthread.php?tid=5420472.*, 2016.

[20] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 2017.

[21] Appneta. Tcpreplay - pcap editing and replaying utilities, 2016.

[22] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*, 2014.

[23] N Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. Seda: Scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 964–975. ACM, 2015.

[24] Johannes Bader. Domain generation algorithm. *https://www.johannesbader.ch/2015/05/the-dga-of-ranbyus/*, 2015.

[25] Jinrong Bai, Yanrong Yang, Shiguang Mu, and Yu Ma. Malware detection through mining symbol table of linux executables. *Information Technology Journal*, 12:380–384, 02 2013.

[26] Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. *arXiv preprint arXiv:1610.00502*, 2016.

[27] Davide Balzarotti, Marco Cova, Christoph Karlberger, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Efficient detection of split personalities in malware. In *NDSS*. Citeseer, 2010.

[28] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, behavior-based malware clustering. In *NDSS*, volume 9, pages 8–11, 2009.

[29] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. *TTAnalyze: A tool for analyzing malware.* na, 2006.

[30] Ulrich Bayer, Andreas Moser, Christopher Kruegel, and Engin Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2(1):67–77, 2006.

[31] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[32] Alex Berry, Josh Homan, and Randi Eitzman. Wannacry malware profile - fireeye, May 23 2017.

[33] Elisa Bertino and Nayeem Islam. Botnets and internet of things security. *Computer*, 50(2):76–79, 2017.

[34] Randeep Bhatia, Steven Benno, Jairo Esteban, TV Lakshman, and John Grogan. 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*, pages 42–48, 2019.

[35] Hugo LJ Bijmans, Tim M Booij, and Christian Doerr. 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*, pages 449–464, 2019.

[36] Olivier Bilodeau and Thomas Dupuy. Dissecting Linux/Moose The Analysis of a Linux Router-based Worm Hungry for Social Networks. (May), 2015.

[37] James R Binkley and Suresh Singh. An algorithm for anomaly-based botnet detection. *SRUTI*, 6:7–7, 2006.

[38] Jean-Marie Borello and Ludovic Mé. Code obfuscation techniques for metamorphic viruses. *Journal in Computer Virology*, 4(3):211–220, 2008.

[39] S Buehlmann and C Liebchen. Joebox: a secure sandbox application for windows to analyse the behaviour of malware, 2010.

[40] Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 621–634. ACM, 2009.

[41] Juan Caballero, Heng Yin, Zhenkai Liang, and Dawn Song. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 317–329. ACM, 2007.

[42] Chen Cao, Le Guan, Peng Liu, Neng Gao, Jingqiang Lin, and Ji Xiang. Hey, you, keep away from my device: remotely implanting a virus expeller to defeat mirai on iot devices. *arXiv preprint arXiv:1706.05779*, 2017.

[43] FO Cetin, C Hernandez Ganan, EM Altena, Takahiro Kasama, Daisuke Inoue, Kazuki Tamiya, Ying Tie, Katsunari Yoshioka, and MJG van Eeten. Cleaning up the internet of evil things: Real-world evidence on isp and consumer efforts to remove mirai. 2019.

[44] Danai Chasaki. *Security issues in networked embedded devices.* University of Massachusetts Amherst, 2012.

[45] Danai Chasaki and Tilman Wolf. Attacks and defenses in the data plane of networks. *IEEE Transactions on dependable and secure computing*, 9(6):798–810, 2012.

[46] Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards automated dynamic analysis for linux-based embedded firmware. In *NDSS*, 2016.

[47] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, XiaoFeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing. In *NDSS*, 2018.

[48] Xu Chen, Jon Andersen, Z Morley Mao, Michael Bailey, and Jose Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 177–186. IEEE, 2008.

[49] Ryan Chinn. Botnet detection: Honeypots and the internet of things. *Unpublished doctoral dissertation). University of Arizona*, 2015.

[50] K Cho, K Mitsuya, and A Kato. Traffic data repository maintained by the mawi working group of the wide project. *URL http://mawi. wide. ad. jp/mawi*, 2005.

[51] Kenjiro Cho, Koushirou Mitsuya, and Akira Kato. Traffic data repository at the wide project. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '00, pages 51–51, Berkeley, CA, USA, 2000. USENIX Association.

[52] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. Technical report, Wisconsin Univ-Madison Dept of Computer Sciences, 2006.

[53] Mihai Christodorescu, Somesh Jha, and Christopher Kruegel. Mining specifications of malicious behavior. In *Proceedings of the 1st India software engineering conference*, pages 5–14. ACM, 2008.

[54] Evan Cooke, Farnam Jahanian, and Danny McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. *SRUTI*, 5:6–6, 2005.

[55] Andrei Costin, Jonas Zaddach, Aurélien Francillon, Davide Balzarotti, and Sophia Antipolis. A large-scale analysis of the security of embedded firmwares. In *USENIX Security Symposium*, pages 95–110, 2014.

[56] Andrei Costin, Apostolis Zarras, and Aurélien Francillon. 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*, pages 437–448. ACM, 2016.

[57] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. Understanding linux malware. In *IEEE Symposium on Security & Privacy*, 2018.

[58] Ang Cui, Jatin Kataria, and Salvatore J Stofo. From prey to hunter: transforming legacy embedded devices into exploitation sensor grids. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 393–402. ACM, 2011.

[59] Ang Cui and Salvatore J Stolfo. A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 97–106. ACM, 2010.

[60] Weidong Cui, Marcus Peinado, Karl Chen, Helen J Wang, and Luis Irun-Briz. Tupni: Automatic reverse engineering of input formats. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 391–402. ACM, 2008.

[61] Fan Dang, Zhenhua Li, Yunhao Liu, Ennan Zhai, Qi Alfred Chen, Tianyin Xu, Yan Chen, and Jingyu Yang. Understanding fileless attacks on linux-based iot devices with honeycloud. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 482–493, 2019.

[62] Ahmad Darki, Chun-Yu Chuang, Michalis Faloutsos, Zhiyun Qian, and Heng Yin. Rare: A systematic augmented router emulation for malware analysis. In *International Conference on Passive and Active Network Measurement*, pages 60–72. Springer, 2018.

[63] Drew Davidson, Benjamin Moench, Thomas Ristenpart, and Somesh Jha. Fie on firmware: Finding vulnerabilities in embedded systems using symbolic execution. In *USENIX Security Symposium*, pages 463–478, 2013.

[64] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. On the feasibility of online malware detection with performance counters. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 559–570. ACM, 2013.

[65] Detux. Github. The Multiplatform Linux Sandbox, May 13 2015.

[66] Christian J Dietrich, Christian Rossow, Felix C Freiling, Herbert Bos, Maarten Van Steen, and Norbert Pohlmann. On botnets that use dns for command and control. In *2011 seventh european conference on computer network defense*, pages 9–16. IEEE, 2011.

[67] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 51–62. ACM, 2008.

[68] Brendan Dolan-Gavitt, Tim Leek, Josh Hodosh, and Wenke Lee. Tappan zee (north) bridge: mining memory accesses for introspection. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 839–850. ACM, 2013.

[69] Brendan F Dolan-Gavitt, Josh Hodosh, Patrick Hulin, Tim Leek, and Ryan Whelan. Repeatable reverse engineering for the greater good with panda. 2014.

[70] Lukas Durfina, Jakub Kroustek, and Petr Zemek. Psybot malware: A step-by-step decompilation case study. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 449–456. IEEE, 2013.

[71] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014.

[72] F-SECURE. Iot threat landscap:e old hacks, new devices. *https://bit.ly/2VmIFSL*, 2019.

[73] Brown Farinholt, Mohammad Rezaeirad, Paul Pearce, Hitesh Dharmdasani, Haikuo Yin, Stevens Le Blond, Damon McCoy, and Kirill Levchenko. 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)*, pages 770–787. Ieee, 2017.

[74] 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.

[75] Qian Feng, Minghua Wang, Mu Zhang, Rundong Zhou, Andrew Henderson, and Heng Yin. Extracting conditional formulas for cross-platform bug search. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 346–359. ACM, 2017.

[76] Qian Feng, Rundong Zhou, Chengcheng Xu, Yao Cheng, Brian Testa, and Heng Yin. Scalable graph-based bug search for firmware images. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 480–491. ACM, 2016.

[77] Tal Garfinkel, Keith Adams, Andrew Warfield, and Jason Franklin. Compatibility is not transparency: Vmm detection myths and realities. In *HotOS*, 2007.

[78] Ioannis Gasparis, Zhiyun Qian, Chengyu Song, and Srikanth V. Krishnamurthy. Detecting android root exploits by learning from root providers. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1129–1144, Vancouver, BC, 2017. USENIX Association.

[79] ROOTKITER Genshen Ye, Hui Wang. The new developments Of the FBot, 2019. `https://blog.netlab.360.com/the-new-developments-of-the-fbot-en/`.

[80] GNU. Gnu binutils. *https://www.gnu.org/software/binutils/*, 2019.

[81] Laurence Goasduff. 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-io*, 2019.

[82] Mikhail Gorobets, Oleksandr Bazhaniuk, Alex Matrosov, Andrew Furtak, and Yuriy Bulygin. Attacking hypervisors via firmware and hardware. *blackhat USA 2015*, 2015.

[83] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, et al. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *USENIX security symposium*, volume 5, pages 139–154, 2008.

[84] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *USENIX Security Symposium*, volume 7, pages 1–16, 2007.

[85] Guofei Gu, Junjie Zhang, and Wenke Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *NDSS*, volume 8, pages 1–18, 2008.

[86] Claudio Guarnieri, Allessandro Tanasi, Jurriaan Bremer, and Mark Schloesser. The cuckoo sandbox, 2012.

[87] Juan David Guarnizo, Amit Tambe, Suman Sankar Bhunia, Martín Ochoa, Nils Ole Tippenhauer, Asaf Shabtai, and Yuval Elovici. Siphon: Towards scalable high-interaction physical honeypots. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, pages 57–68. ACM, 2017.

[88] Eric Gustafson, Marius Muench, Chad Spensky, Nilo Redini, Aravind Machiry, Yanick Fratantonio, Davide Balzarotti, Aurélien Francillon, Yung Ryn Choe, Christophe Kruegel, 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)*, pages 135–150, 2019.

[89] Eric Gustafson, Marius Muench, Chad Spensky, Nilo Redini, Aravind Machiry, Yanick Fratantonio, Davide Balzarotti, Aurélien Francillon, Yung Ryn Choe, Christophe Kruegel, 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)*, pages 135–150, 2019.

[90] Nikolai Hampton and Patryk Szewczyk. A survey and method for analysing soho router firmware currency. *Australian Information Security Management Conference*, 2015.

[91] David Heldenbrand and Christopher Carey. The linux router: an inexpensive alternative to commercial routers in the lab. *Journal of Computing Sciences in Colleges*, 23(1):127–133, 2007.

[92] Andrew Henderson, Aravind Prakash, Lok Kwong Yan, Xunchao Hu, Xujiewen Wang, Rundong Zhou, and Heng Yin. Make it work, make it right, make it fast: Building a platform-neutral whole-system dynamic binary analysis platform. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 248–258. ACM, 2014.

[93] Grant Hernandez, Orlando Arias, Daniel Buentello, and Yier Jin. Smart nest thermostat: A smart spy in your home. *Black Hat USA*, 2014.

[94] Stephen Herwig, Katura Harvey, George Hughey, Richard Roberts, and Dave Levin. Measurement and analysis of hajime, a peer-to-peer iot botnet. In *NDSS*, 2019.

[95] SA Hex-Rays. Idapro disassembler, 2008.

[96] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of computer security*, 6(3):151–180, 1998.

[97] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C Freiling. Measuring and detecting fast-flux service networks. In *NDSS*, 2008.

[98] Kekai Hu. Securing network processors with hardware monitors. 2015.

[99] Xin Hu, Tzi-cker Chiueh, and Kang G Shin. Large-scale malware indexing using function-call graphs. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 611–620. ACM, 2009.

[100] Genshen Ye Hui Wang. Emptiness: A New Evolving Botnet, 2019. `https://blog.netlab.360.com/emptiness-a-new-evolving-botnet/`.

[101] Nwokedi Idika and Aditya P Mathur. A survey of malware detection techniques. *Purdue University*, 48, 2007.

[102] INTERPOL. 'internet of things' cyber risks tackled during interpol digital security challenge. *https://bit.ly/2LIeIwR*, 2018.

[103] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[104] Min Gyung Kang, Heng Yin, Steve Hanna, Stephen McCamant, and Dawn Song. Emulating emulation-resistant malware. In *Proceedings of the 1st ACM workshop on Virtual machine security*, pages 11–22, 2009.

[105] Joris Kinable and Orestis Kostakis. Malware classification based on call graph clustering. *Journal in computer virology*, 7(4):233–245, 2011.

[106] Dhilung Kirat and Giovanni Vigna. Malgene: Automatic extraction of malware analysis evasion signature. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 769–780. ACM, 2015.

[107] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. Barecloud: bare-metal analysis-based evasive malware detection. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 287–301, 2014.

[108] Akos Kiss, Judit Jász, Gábor Lehotai, and Tibor Gyimóthy. Interprocedural static slicing of binary executables. In *Source Code Analysis and Manipulation, 2003. Proceedings. Third IEEE International Workshop on*, pages 118–127. IEEE, 2003.

[109] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiao-yong Zhou, and XiaoFeng Wang. Effective and efficient malware detection at the end host. In *USENIX security symposium*, pages 351–366, 2009.

[110] Clemens Kolbitsch, Thorsten Holz, Christopher Kruegel, and Engin Kirda. Inspector gadget: Automated extraction of proprietary gadgets from malware binaries. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 29–44. IEEE, 2010.

[111] Maryna Kolisnyk, Vyacheslav Kharchenko, Iryna Piskachova, and Nikolaos Bardis. A markov model of iot system availability considering ddos attacks and energy modes of server and router.

[112] Jesse Kornblum. Fuzzy hashing and ssdeep, 2010.

[113] Christian Kreibich, Nicholas Weaver, Weidong Cui, Vern Paxson, and Chris Kanich. Gq: Practical containment for measuring modern malware systems.

[114] Bum Jun Kwon, Jayanta Mondal, Jiyong Jang, Leyla Bilge, and Tudor Dumitras. The dropper effect: Insights into malware distribution with downloader graph analytics. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1118–1129. ACM, 2015.

[115] Andrea Lanzi, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, and Engin Kirda. Accessminer: using system-centric models for malware protection. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 399–412. ACM, 2010.

[116] Boris Lau and Vanja Svajcer. Measuring virtual machine detection in malware using dsd tracer. *Journal in Computer Virology*, 6(3):181–195, 2010.

[117] Corrado Leita and Marc Dacier. Sgnet: a worldwide deployable framework to support the analysis of malware threat models. In *2008 Seventh European Dependable Computing Conference*, pages 99–109. IEEE, 2008.

[118] Corrado Leita, Ken Mermoud, and Marc Dacier. Scriptgen: an automated script generation tool for honeyd. In *21st Annual Computer Security Applications Conference (ACSAC'05)*, pages 12–pp. IEEE, 2005.

[119] Chaz Lever, Platon Kotzias, Davide Balzarotti, Juan Caballero, and Manos Antonakakis. A lustrum of malware network communication: Evolution and insights. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 788–804. IEEE, 2017.

[120] Limon. Github. Sandbox for Analyzing Linux Malwares, May 13 2015.

[121] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. Detecting environment-sensitive malware. In *International Workshop on Recent Advances in Intrusion Detection*, pages 338–357. Springer, 2011.

[122] Eduardo Novella Lorente, Carlo Meijer, and Roel Verdult. Scrutinizing wpa2 password generating algorithms in wireless routers. In *WOOT*, 2015.

[123] Kangjie Lu, Zhichun Li, Vasileios P Kemerlis, Zhenyu Wu, Long Lu, Cong Zheng, Zhiyun Qian, Wenke Lee, and Guofei Jiang. Checking more and alerting less: Detecting privacy leakages via enhanced data-flow analysis and peer voting. In *NDSS*, 2015.

[124] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. Chex: statically vetting android apps for component hijacking vulnerabilities. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 229–240. ACM, 2012.

[125] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang. Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices. In *BlackHat Las Vegas, NV 2017*. BlackHat, 2017.

[126] Kin-Keung Ma, Khoo Yit Phang, Jeffrey Foster, and Michael Hicks. Directed symbolic execution. *Static Analysis*, pages 95–111, 2011.

[127] Thomas Mandl, Ulrich Bayer, and Florian Nentwich. Anubis-analyzing unknown binaries the automatic way. 2009.

[128] Armel Mangean, Jean-Luc Béchennec, Mikaël Briday, and Sébastien Faucou. Best: a binary executable slicing tool. In *OASIcs-OpenAccess Series in Informatics*, volume 55. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[129] Rokon Md Omar Faruk, Risul Islam, Ahmad Darki, Vagelis E. Papalexakis, and Michalis Faloutsos. Sourcefinder: Finding malware source-code from publicly available repositories. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020)*, 2020.

[130] MITRE. Mitre att&ck. *https://attack.mitre.org/*, 2019.

[131] Alper T Mizrak, Stefan Savage, and Keith Marzullo. Detecting compromised routers via packet forwarding behavior. *IEEE network*, 22(2), 2008.

[132] Andreas Moser, Christopher Kruegel, and Engin Kirda. Exploring multiple execution paths for malware analysis. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 231–245. IEEE, 2007.

[133] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*, pages 421–430. IEEE, 2007.

[134] Kamini Nalavade. Intrusion detection and prevention system for routing protocol attacks.

[135] NationalSecurityAgency. Ghidra is a software reverse engineering (sre) framework. *https://www.nsa.gov/resources/everyone/ghidra/*, 2019.

[136] Muhammad Naveed, Shams un Nihar, and Mohammad Inayatullah Babar. Network intrusion prevention by configuring acls on the routers, based on snort ids alerts. In *Emerging Technologies (ICET), 2010 6th International Conference on*, pages 234–239. IEEE, 2010.

[137] Marcin Nawrocki, Matthias Wählisch, Thomas C Schmidt, Christian Keil, and Jochen Schönfelder. A survey on honeypot software and data analysis. *arXiv preprint arXiv:1608.06249*, 2016.

[138] Dang Tu Nguyen, Chengyu Song, Zhiyun Qian, Srikanth V Krishnamurthy, Edward JM Colbert, and Patrick McDaniel. Iotsan: Fortifying the safety of iot systems. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, pages 191–203, 2018.

[139] Marcus Niemietz and Jörg Schwenk. Owning your home network: Router security revisited. *arXiv preprint arXiv:1506.04112*, 2015.

[140] NIST. CVE-2016-10372. Available from MITRE., May 16 2016.

[141] NIST. CVE-2017-3882. Available from MITRE., May 16 2017.

[142] NIST. CVE-2017-8225. Available from MITRE., 2017.

[143] NIST. CVE-2018-10561. Available from MITRE., May 3 2018.

[144] NIST. CVE-2018-6530. Available from MITRE., 2018.

[145] Michel Oosterhof. Cowrie ssh/telnet honeypot. *https://github.com/cowrie/cowrie*, 2014.

[146] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot: Analysing the rise of iot compromises. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., 2015. USENIX Association.

[147] Roberto Paleari, Lorenzo Martignoni, Giampaolo Fresi Roglia, and Danilo Bruschi. A fistful of red-pills: How to automatically generate procedures to detect cpu emulators. In *Proceedings of the USENIX Workshop on Offensive Technologies (WOOT)*, volume 41, page 86, 2009.

[148] Dorottya Papp, Zhendong Ma, and Levente Buttyan. Embedded systems security: Threats, vulnerabilities, and attack taxonomy. In *Privacy, Security and Trust (PST), 2015 13th Annual Conference on*, pages 145–152. IEEE, 2015.

[149] Masarah Paquet-Clouston, Olivier Bilodeau, and David Décary-Hétu. Can we trust social media data?: Social network manipulation by an iot botnet. In *Proceedings of the 8th International Conference on Social Media & Society*, page 15. ACM, 2017.

[150] Sri Parameswaran and Tilman Wolf. Embedded systems security—an overview. *Design Automation for Embedded Systems*, 12(3):173–183, 2008.

[151] Younghee Park, Douglas Reeves, Vikram Mulukutla, and Balaji Sundaravel. Fast malware classification by automated behavioral graph matching. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, page 45. ACM, 2010.

[152] Fei Peng, Zhui Deng, Xiangyu Zhang, Dongyan Xu, Zhiqiang Lin, and Zhendong Su. X-force: Force-executing binary programs for security applications. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 829–844, 2014.

[153] Jannik Pewny, Behrad Garmany, Robert Gawlik, Christian Rossow, and Thorsten Holz. Cross-architecture bug search in binary executables. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 709–724. IEEE, 2015.

[154] Radare. Reverse engineering framework and commandline tools. *https://github.com/radare/radare2*, 2019.

[155] Siegfried Rasthofer, Steven Arzt, Marc Miltenberger, and Eric Bodden. Harvesting runtime values in android applications that feature anti-analysis techniques. In *NDSS*, 2016.

[156] John W Ratcliff and David E Metzener. Pattern-matching-the gestalt approach. *Dr Dobbs Journal*, 13(7):46, 1988.

[157] Jingjing Ren, Daniel J Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach. In *Proceedings of the Internet Measurement Conference*, pages 267–279, 2019.

[158] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125. Springer, 2008.

[159] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668, 2011.

[160] B Rodrigues. Luabot: Malware targeting cable modems, 2016.

[161] Christian Rossow, Christian J Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten Van Steen, Felix C Freiling, and Norbert Pohlmann. Sandnet: Network traffic analysis of malicious software. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pages 78–88. ACM, 2011.

[162] Christian Rossow, Christian J Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten Van Steen, Felix C Freiling, and Norbert Pohlmann. Sandnet: Network traffic analysis of malicious software. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pages 78–88. ACM, 2011.

[163] Kevin Roundy and Barton Miller. Hybrid analysis and control of malware. In *Recent Advances in Intrusion Detection*, pages 317–338. Springer, 2010.

[164] Michal Sajdak. Remote root shell on a soho class router, 2009.

[165] Norman Sandbox. Norman sandbox whitepaper, 2010.

[166] Eric M Schulte, Westley Weimer, and Stephanie Forrest. Repairing cots router firmware without access to source code or test suites: A case study in evolutionary software repair. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 847–854. ACM, 2015.

[167] Farrukh Shahzad and Muddassar Farooq. Elf-miner: Using structural knowledge and data mining methods to detect new (linux) malicious executables. *Knowl. Inf. Syst.*, 30(3):589–612, March 2012.

[168] Sri Shaila G, Ahmad Darki, Michalis Faloutsos, Nael Abu-Ghazaleh, Manu Sridharan, et al. Idapro for iot malware analysis? In *12th {USENIX} Workshop on Cyber Security Experimentation and Test ({CSET} 19)*, 2019.

[169] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware. In *NDSS*, 2015.

[170] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *IEEE Symposium on Security and Privacy*, 2016.

[171] Amjad Basha M Sikiligiri. *Buffer overflow attack and prevention for embedded systems.* PhD thesis, University of Cincinnati, 2011.

[172] Bilhanan Silverajan, Markku Vajaranta, and Antti Kolehmainen. Home network security: Modelling power consumption to detect and prevent attacks on homenet routers. In *Information Security (AsiaJCIS), 2016 11th Asia Joint Conference on*, pages 9–16. IEEE, 2016.

[173] Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena. Bitblaze: A new approach to computer security via binary analysis. *Information systems security*, pages 1–25, 2008.

[174] Lance Spitzner. *Honeypots: tracking hackers*, volume 1. Addison-Wesley Reading, 2003.

[175] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647. ACM, 2009.

[176] Seyed Mohammadjavad Seyed Talebi, Hamid Tavakoli, Hang Zhang, Zheng Zhang, Ardalan Amiri Sani, and Zhiyun Qian. Charm: Facilitating dynamic analysis of device drivers of mobile systems. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 291–307, 2018.

[177] Cisco Talos. New vpnfilter malware targets at least 500 k networking devices worldwide, 2019.

[178] Kimberly Tam, Salahuddin J Khan, Aristide Fattori, and Lorenzo Cavallaro. Copperdroid: Automatic reconstruction of android malware behaviors. In *NDSS*, 2015.

[179] TheWhiteTeam. Linux.wifatch. *https://gitlab.com/rav7teif/linux.wifatch/project_members*, 2019.

[180] Ronghua Tian, Lynn Margaret Batten, and SC Versteeg. Function length as a tool for malware classification. In *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, pages 69–76. IEEE, 2008.

[181] Virus Total. Virustotal-free online virus, malware and url scanner. *Online: https://www. virustotal. com/en*, 2012.

[182] Johannes B. Ullrich. Linksys worm ”themoon” summary: What we know so far. *goo.gl/giHF7Q*, 2014.

[183] US-CERT. US Computer Emergency Readiness Team heightened ddos threat posed by mirai and other botnets, alert ta16-288a. Accessed: 2016-11-30.

[184] Rob van der Meulen. Gartner says 8.4 billion connected “things” will be in use in 2017, up 31 percent from 2016, gartner. *Online: https://www.gartner.com/newsroom/id/3598917*, 2017.

[185] Amit Vasudevan and Ramesh Yerraballi. Cobra: Fine-grained malware analysis using stealth localized-executions. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.

[186] Pierre-Antoine Vervier and Yun Shen. Before toasters rise up: A view into the emerging iot threat landscape. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 556–576. Springer, 2018.

[187] Timothy Vidas and Nicolas Christin. Evading android runtime analysis via sandbox detection. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 447–458. ACM, 2014.

[188] Saili Waichal and BB Meshram. Router attacks-detection and defense mechanisms. *Int. J. Sci. Technol. Res*, 2:145–149, 2013.

[189] Hui Wang and RootKiter. Bcmpupnp_hunter: A 100k botnet turns home routers to email spammers. *https://blog.netlab.360.com/bcmpupnp_hunter-a-100k-botnet-turns-home-routers-to-email-spammers-en/*, 2018.

[190] Ping Wang, Lei Wu, Baber Aslam, and Cliff C Zou. A systematic study on peer-to-peer botnets. In *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*, pages 1–8. IEEE, 2009.

[191] Rootkiter Wang, Hui and Yegenshen. Gpon exploit in the wild (iv) - themoon botnet join in with a 0day(?). *https://blog.netlab.360.com/gpon-exploit-in-the-wild-iv-themoon-botnet-join-in-with-a-0day/*, 2014.

[192] Xueyang Wang, Charalambos Konstantinou, Michail Maniatakos, Ramesh Karri, Serena Lee, Patricia Robison, Paul Stergiou, and Steve Kim. Malicious firmware detection with hardware performance counters. *IEEE Transactions on Multi-Scale Computing Systems*, 2(3):160–173, 2016.

[193] David Watson and Jamie Riden. The honeynet project: Data collection tools, infrastructure, archives and analysis. In *2008 WOMBAT Workshop on Information Security Threats Data Collection and Sharing*, pages 24–30. IEEE, 2008.

[194] Wikipedia. October 2016 dyn cyberattack, 2016.

[195] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, 5(2), 2007.

[196] Nigel Williams and Sebastian Zander. Real time traffic classification and prioritisation on a home router using diffuse. 2012.

[197] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. 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*, pages 363–376. ACM, 2017.

[198] Zhaoyan Xu, Antonio Nappa, Robert Baykov, Guangliang Yang, Juan Caballero, and Guofei Gu. Autoprobe: Towards automatic active malicious server probing using dynamic binary analysis. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 179–190. ACM, 2014.

[199] Zhaoyan Xu, Jialong Zhang, Guofei Gu, and Zhiqiang Lin. Goldeneye: Efficiently and effectively unveiling malware's targeted environment. In *International Workshop on Recent Advances in Intrusion Detection*, pages 22–45. Springer, 2014.

[200] Lun-Pin Yuan, Wenjun Hu, Ting Yu, Peng Liu, and Sencun Zhu. Towards large-scale hunting for android negative-day malware. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, pages 533–545, 2019.

[201] Insu Yun, Sangho Lee, Meng Xu, Yeongjin Jang, and Taesoo Kim. {QSYM}: A practical concolic execution engine tailored for hybrid fuzzing. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 745–761, 2018.

[202] Jonas Zaddach, Luca Bruno, Aurelien Francillon, and Davide Balzarotti. Avatar: A framework to support dynamic security analysis of embedded systems' firmwares. In *NDSS*, 2014.

[203] Xiangyu Zhang and Rajiv Gupta. Matching execution histories of program versions. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 197–206. ACM, 2005.

[204] Yaowen Zheng, Ali Davanian, Heng Yin, Chengyu Song, Hongsong Zhu, and Limin Sun. Firm-afl: high-throughput greybox fuzzing of iot firmware via augmented process emulation. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1099–1114, 2019.