

**UNIVERSITY OF CASTILLA-LA MANCHA**

**Computing Systems Department**



**Modeling, Analysis, and Detection of Threats  
in the Internet-Of-Things Ecosystem**

A dissertation for the degree of Doctor of Philosophy in Computer Science  
to be presented with due permission of the Computing Systems  
Department, for public examination and debate.

Author: D. Javier Carrillo Mondéjar

Advisor: Dr. D. José Luis Martínez Martínez  
Dr. D. Guillermo Suarez de Tangil

**Albacete, November 2022**



**UNIVERSIDAD DE CASTILLA-LA MANCHA**

**Departamento de Sistemas Informáticos**



## **Modeling, Analysis, and Detection of Threats in the Internet-Of-Things Ecosystem**

Tesis Doctoral presentada al Departamento de Sistemas Informáticos de la  
Universidad de Castilla-La Mancha para la obtención del título de Doctor  
en Tecnologías Informáticas Avanzadas.

Autor: D. Javier Carrillo Mondéjar  
Director: Dr. D. José Luis Martínez Martínez  
Dr. D. Guillermo Suarez de Tangil

**Albacete, Noviembre de 2022**



*A mi familia...*  
*por confiar siempre en mí.*



# Agradecimientos

En primer lugar, me gustaría agradecer a mis dos directores, José Luis y Guillermo, por su esfuerzo y dedicación a lo largo de la tesis. Sin vosotros esto no habría sido posible. Me habéis guiado durante todos estos años, dándome la libertad y autonomía para ir creciendo como investigador y dedicarme a lo que me gusta durante estos años. Habéis estado siempre dispuestos a brindarme vuestra ayuda, encontrando soluciones y repuestas para cualquier duda o problema que me pudiera surgir a lo largo de todo este tiempo. Agradezco que os hayáis tomado siempre el tiempo para aconsejarme sobre decisiones que tenían que ver sobre mi carrera académica y profesional.

A todo el profesorado de la Escuela Superior de Ingeniería Informática y del Instituto de Investigación en Informática, que durante estos años nos han transmitido su conocimiento y que de un modo u otro aportaron su granito de arena para que podamos conseguir objetivos como éste. Me gustaría destacar a Pedro Cuenca, por su cercanía, consejos y disposición a prestarnos su ayuda siempre que la hemos necesitado.

A toda la gente de la Universidad de Jyväskylä y King's College London que me acogió durante mis estancias de investigación. En especial me gustaría agradecer tanto a Andrei como a Guillermo el trato que recibí durante esos meses. Ha sido un placer haber podido trabajar con vosotros durante este tiempo y espero que podamos seguir colaborando en un futuro próximo.

A mi familia, porque es muy fácil caminar sin mirar atrás cuando tienes gente cerca que te va a ayudar de manera incondicional, y que van a estar siempre ahí para los éxitos, pero sobre todo para los fracasos. Ninguna palabra que pueda expresar en estas líneas será suficiente para agradecer todo lo que habéis hecho por mí. Siempre me habéis apoyado en cualquiera de las decisiones que he tomado, aún incluso cuando no estuvierais completamente de acuerdo. A mis padres, porque soy quien soy gracias a vuestro esfuerzo y sacrificio. A mi hermano, por ser un modelo a seguir para mí y cuya presencia siempre me transmite tranquilidad. A mi cuñada, por estar siempre bromeando y hacer que no nos tomemos todo tan en serio. A mis sobrinos, por sacar lo mejor de mí y hacerme reír siempre con vuestras acciones. Por sus "¿Has terminado ya tus *deberes*?, ¿Te vienes a jugar con nosotros?, Javiiiiier, vamos, vamos", que siempre consiguen hacer que desconecte cuando lo necesito.

A mis amigos, por estar siempre ahí, ya sea para tomar un café y charlar, pasear o salir a tomar algo para pasar un rato juntos y desconectar. Aunque me encantaría poder nombraros a todos, por suerte sois demasiados y me sería imposible. Si que me gustaría destacar a amigos que han recorrido el mismo camino, y con los que he luchado para conseguir los mismos objetivos. Muchos de nosotros empezamos siendo compañeros en algún trabajo del grado o simplemente de clase, pero congeniamos rápidamente y nos volvimos inseparables durante toda esta etapa. Otros entraron en nuestras vidas más tarde, pero parece que han estado ahí desde el principio. David, Juanma, Rubén, Carlos, José y Sergio habéis conseguido que esta etapa haya sido más llevadera, amenizando las horas en el laboratorio y consiguiendo que los días de trabajo en el confinamiento fueran menos duros. Muchas gracias por todo, a vuestro lado todo parece más fácil. Sois muy grandes y conseguiréis todo lo que os propongáis.

---

Este trabajo ha sido cofinanciado por el Ministerio de Economía y Competitividad y la Comisión Europea (fondos MINECO/FEDER), bajo el proyecto con referencia RTI2018-098156-B-C52, por la Consejería de Educación, Cultura y Deportes de la Junta de Comunidades de Castilla-La Mancha, mediante el proyecto con referencia SBPLY/17/180501/000353, y por el Ministerio de Educación, Cultura y Deporte a través de la beca FPU 17/03105.



# Summary

The appearance of the Internet of Things (IoT) in the technological world has brought about a revolution when it comes to integrating technology into the most everyday aspects of our lives. While traditional technological scenarios require direct interaction with devices, such as a computer or a mobile phone, the IoT world bases much of its logic on being transparent to the user while providing functionality. We find immediate examples in smart assistants, sensors, smart watches and video surveillance cameras. In addition, as if this were not enough, its application in certain areas has given rise to new technological scenarios such as smart cities, smart homes, cyber medicine and Industry 5.0.

Although the success of IoT is undoubted and has changed the way users interact with technology, it brings with it great concerns regarding user security and privacy. The IoT ecosystem is made up of a large number of networked devices that interact with each other using a wide range of communication protocols. In general, these devices are designed to offer innovative services at a low price, but not much attention has been paid to security in their development. This, together with the inherent characteristics of this type of device, such as limited computing capacity and storage, as well as lack of updates from manufacturers, have led to these devices often presenting major security flaws that range from the use of default passwords or configurations to outdated software that contains publicly-known security vulnerabilities. This has caused the IoT to become one of the main targets for cybercriminals to attack, and the number of samples of malware specially designed for the IoT has grown significantly in recent years.

In view of all the above, this thesis analyzes the problem of security in the IoT ecosystem with the aim of modeling, analyzing and detecting the main threats. First, we explore the SSH, Telnet and VoIP protocols through honeypot systems to analyze and understand how these protocols are exploited by cybercriminals and what actions they carry out. Second, we design and develop a framework to address the problem of Linux-based malware that is specially aimed at IoT environments, allowing the analysis and extraction of malware features both statically and dynamically, as well as a comparison with other samples that have already been studied. Then, we propose a methodology for automating the process of relating malware to known threats and vetting new unknown samples. Finally, we propose a methodology for adding defensive capabilities to IoT devices that are abandoned and no longer maintained by the manufacturer, allowing the addition of an additional layer of security against known threats in order to protect the devices against such attacks.



# Resumen

La aparición del Internet de las Cosas (IoT) en el mundo tecnológico ha supuesto una revolución a la hora de integrar la tecnología en los aspectos más cotidianos de nuestra vida. Mientras que los escenarios tecnológicos tradicionales requieren de una interacción directa con los dispositivos, como pueden ser un ordenador o un teléfono móvil, el mundo IoT basa gran parte de su lógica en ser transparente al usuario al mismo tiempo que proporciona funcionalidad. Encontramos ejemplos inmediatos en asistentes inteligentes, sensores, relojes inteligentes o cámaras de videovigilancia. Además, por si esto no fuera poco, su aplicación en ciertos ámbitos ha dado lugar a la aparición de nuevos escenarios tecnológicos como las ciudades inteligentes, hogares inteligentes, cibermedicina o la Industria 5.0.

Aunque el éxito de IoT es indudable y ha cambiado la forma en la que los usuarios interaccionan con la tecnología, trae consigo grandes preocupaciones en cuanto a la seguridad y privacidad de los usuarios. El ecosistema del IoT está compuesto por un gran número de dispositivos conectados en la red que interactúan entre sí utilizando un amplio abanico de protocolos de comunicación. En general, estos dispositivos están pensados para ofrecer servicios innovadores a bajo precio y no desde el punto de vista de la seguridad. Esto junto a las características inherentes a este tipo de dispositivos como la baja capacidad de cómputo, almacenamiento y falta de actualizaciones por parte de los fabricantes, han propiciado que estos dispositivos a menudo presenten grandes problemas de seguridad que van desde el uso de contraseñas o configuraciones por defecto a software desactualizado que contiene problemas de seguridad públicamente conocidos en Internet. Esto ha favorecido que IoT se haya convertido en uno de los principales entornos a atacar por los cibercriminales y que la cantidad de muestras de *malware* especialmente diseñado para IoT haya crecido en los últimos años.

Por todo lo expuesto anteriormente, esta Tesis analiza el problema de la seguridad en el ecosistema IoT con el objetivo de modelar, analizar y detectar las principales amenazas. En primer lugar, se exploran los protocolos SSH, Telnet y VoIP a través de sistemas señuelo para analizar y comprender como estos protocolos son abusados por los cibercriminales y que acciones llevan a cabo. En segundo lugar, se diseña e implementa un *framework* para abordar el problema del *malware* basado en Linux especialmente dirigido a entornos IoT, permitiendo analizar y extraer características del malware tanto de manera estática como dinámica, así como el grado de similitud con otras muestras ya exploradas. Después, se propone una

metodología para automatizar el proceso de relacionar *malware* a amenazas conocidas e investigación de nuevas muestras desconocidas. Por último, se propone una metodología para añadir capacidades defensivas a dispositivos IoT que se encuentran abandonados y ya no son mantenidos por el fabricante, permitiendo añadir una capa adicional de seguridad ante amenazas conocidas con el fin de proteger los dispositivos ante estos ataques.

# Contents

<b>Contents</b>	<b>vii</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Acronyms</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Justification . . . . .	1
1.2 Objectives . . . . .	6
1.3 Methodology and Work Plan . . . . .	7
1.4 General Discussion and Description of the Proposals . . . . .	10
1.4.1 Measurement of attack trends . . . . .	11
1.4.2 Tools and techniques to assist malware analyst . . . . .	14
1.4.3 Defensive measures to protect IoT devices . . . . .	17
1.5 Results . . . . .	19
<b>2 Hajime’s Return: Stories from a Customized Honeypot for the IoT</b>	<b>25</b>
<b>3 On how VoIP attacks foster the malicious call ecosystem</b>	<b>37</b>
<b>4 A chatbot for selectively engaging with online fraudsters</b>	<b>53</b>
<b>5 Automatic Analysis Architecture of IoT Malware Samples</b>	<b>67</b>
<b>6 Characterizing Linux-based malware: Findings and recent trends</b>	<b>81</b>
<b>7 HALE-IoT: HARDening LEGacy Intenet-of-Things devices via retrofitting defensive firmware modifications and implants</b>	<b>97</b>
<b>8 Conclusions and Future Work</b>	<b>119</b>
8.1 Conclusions . . . . .	119
8.2 Future Work . . . . .	122
<b>Bibliography</b>	<b>125</b>



# List of Figures

1.1	Number of vulnerabilities in IoT devices that have appeared in recent years.	2
1.2	Growth of IoT/Linux malware in recent years. . . . .	3
1.3	Scheme that describes the main objective of this doctoral dissertation. . . .	7
1.4	Methodology followed to evaluate the actions carried out in the VoIP honeypot. . . . .	12
1.5	Methodology that has been followed to identify the type of fraud. . . . .	14
1.6	Scheme of the system architecture. . . . .	14
1.7	Block diagram of the architecture designed. . . . .	15
1.8	Block diagram of the malware sample characterization methodology. . . .	18
1.9	IoT firmware system diagram. . . . .	19
1.10	IoT firmware system diagram AFTER hardening. . . . .	20





# List of Acronyms

<b>API</b>	Application Programming Interfaces
<b>CoAP</b>	Constrained Application Protocol
<b>CPU</b>	Central Processing Unit
<b>DDoS</b>	Distributed Denial-of-Service
<b>ELF</b>	Extensible Linkable Format
<b>EOL</b>	End-Of-Life
<b>FTP</b>	File Transfer Protocol
<b>GGs</b>	GII-GRIN-SCIE
<b>GPT-3</b>	Generative Pre-trained Transformer 3
<b>HALE-IoT</b>	HArdening LEgacy Intenet-of-Things
<b>HNAP</b>	Home Network Administration Protocol
<b>HTTP</b>	HyperText Transfer Protocol
<b>HTTPS</b>	HyperText Transfer Protocol Secure
<b>IEEE</b>	Institute of Electrical and Electronics Engineers
<b>IF</b>	Impact Factor
<b>IoT</b>	Internet of Things
<b>IP</b>	Internet Protocol
<b>ISA</b>	Instruction Set Architecture
<b>JCR</b>	Journal Citation Reports
<b>MITM</b>	Man In The Middle
<b>ML</b>	Machine Learning
<b>MQTT</b>	Message Queue Telemetry Transport

## List of Acronyms

---

<b>NLU</b>	Natural Language Understanding
<b>PBX</b>	Private Branch Exchange
<b>QEMU</b>	Quick EMUlator
<b>RAM</b>	Random Access Memory
<b>REST</b>	Representational State Transfer
<b>SIP</b>	Session Initiation Protocol
<b>SSH</b>	Secure Shell
<b>SSL</b>	Secure Socket Layer
<b>Telnet</b>	Teletype Network
<b>TLS</b>	Transport Layer Security
<b>TOR</b>	The Onion Router
<b>TTP</b>	Tactics, Techniques and Procedures
<b>TV</b>	Television
<b>UPnP</b>	Universal Plug and Play
<b>UPX</b>	Ultimate Packer for eXecutables
<b>URL</b>	Uniform Resource Locator
<b>VoIP</b>	Voice over Internet Protocol
<b>VPN</b>	Virtual Private Network
<b>WAF</b>	Web Application Firewall
<b>XMPP</b>	Extensible Messaging and Presence Protocol

# CHAPTER 1

## Introduction

This chapter introduces the main motivation for this doctoral thesis. In the first place, the situation in the field of security in the Internet of Things (IoT) is outlined, defining the current problems and describing the motivation behind the work carried out. In addition, the objectives of this work are presented together with the methodology employed to achieve them. Finally, a brief description of the proposed solutions is given together with the results obtained in the research process.

### 1.1 Motivation and Justification

In recent years, there has been a sharp increase in the number of devices that are connected to the Internet. This has led to users having access to features and services that have shaped the way they now interact with technology in an unprecedented manner. The integration of technology in environments that differ from the conventional desktop or mobile platforms is what is known as the IoT. This term refers to technology that is connected to the Internet, ranging from sensors, smart televisions (TVs), smart watches, Voice over Internet Protocol (VoIP) phones, and video surveillance cameras, to even refrigerators and vacuum cleaners. IoT devices make people's lives easier by allowing them to perform simple tasks via these devices. For example, nowadays users can perform everyday tasks such as turning the light in their house on or off by using their own voice or opening the door lock remotely.

The impact that IoT has on people's lives is evidenced by the number of IoT devices currently connected to the Internet, which exceeded the number of non-IoT devices for the first time in 2020, with the IoT accounting for 54% of connected devices [1]. The growth in the number of IoT devices seemed to plateau between 2021 and 2022 due to the shortage of chips in the aftermath of the COVID pandemic, but the number of connected devices is expected to reach 27 billion by 2025 [2]. Currently, there is a huge number of devices connected worldwide, with figures reaching about 12.3 billion [2].

## 1.1. Motivation and Justification

---

Although the success of the IoT and the attractive features that it provides to users have changed the way in which technology is interacted with, they bring with them great concerns regarding the security and privacy of user data. The IoT ecosystem is made up of a large number of devices that are interconnected in the network and which use a wide range of protocols to communicate with each other (e.g., Message Queue Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), and Universal Plug and Play (UPnP)). None of these systems are designed from a security standpoint, but from one of cost-effectiveness, offering innovative features to the detriment of security and privacy. And this is in addition to the intrinsic characteristics of the devices, such as low storage capacity, low computing capacity, the lack of software/firmware updates from manufacturers or, even if there are updates, the difficulty of installing them on the devices for even more advanced users.

These characteristics of the IoT make devices vulnerable to different types of attacks. Among the most common vulnerabilities in the IoT are the use of weak or hardcoded credentials, insecure network services exposed directly to the Internet, and the lack of privacy and encryption with sensitive data that the device can handle [3]. Figure 1.1 shows the number of vulnerabilities that have appeared in IoT devices in recent years [4], thus illustrating the magnitude of the problem. This fosters an attractive environment for cybercriminals, causing the number of attacks against this type of device to increase every day [5]. This leads to the fact that the amount of malicious software targeting this environment has grown in recent years, taking advantage of vulnerabilities in the IoT to take control of devices and infect them.

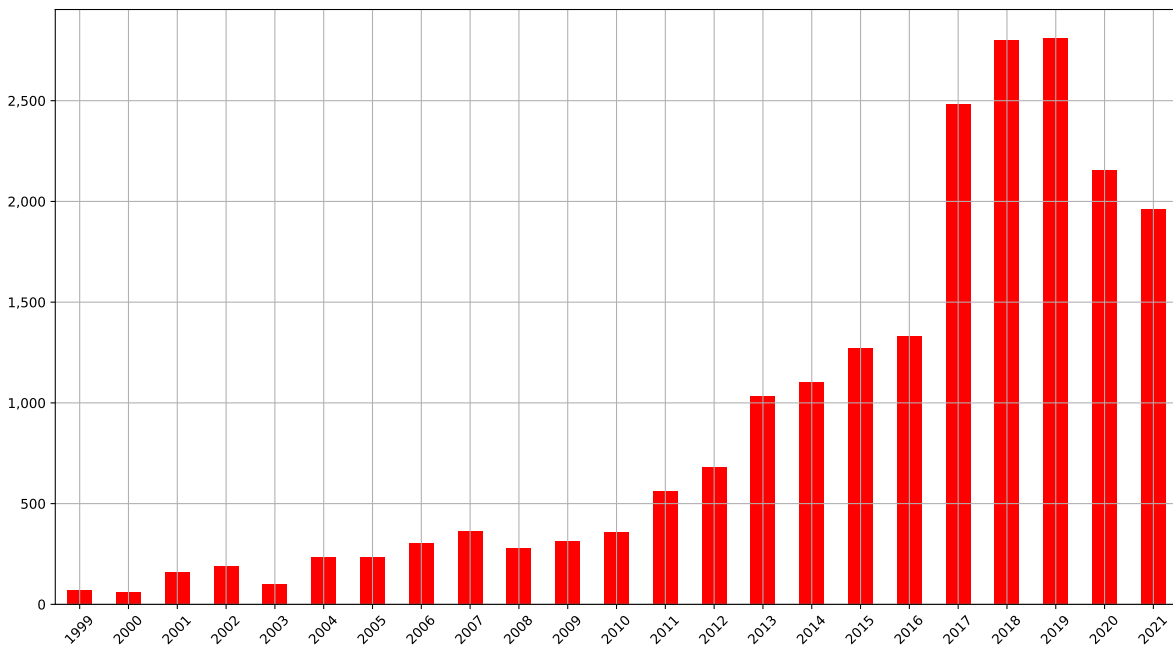


Figure 1.1: Number of vulnerabilities in IoT devices that have appeared in recent years.

The evolution of malware designed to attack IoT devices is relatively new compared with malware targeting desktop systems. The first family of malware specially designed

to attack IoT devices was the Mirai botnet. This botnet infected around 600,000 devices and caused one of the most serious Distributed Denial-of-Service (DDoS) attacks on the Internet [6]. This put the security of IoT devices in the spotlight, since initially Mirai did not use any vulnerability to gain access to the devices, but rather a set of default usernames and passwords to gain access to the devices and install the malware. After the release of the Mirai source code, new malware families have continued to appear. Figure 1.2 shows the cumulative growth in malware samples that targets Linux/IoT Devices from August 2016 to July 2022 [7]. In this graph it can be seen that the number of samples designed for these devices in 2016 was around 67,000, and that currently the number reaches 4,000,000 samples, which shows the magnitude of this problem.

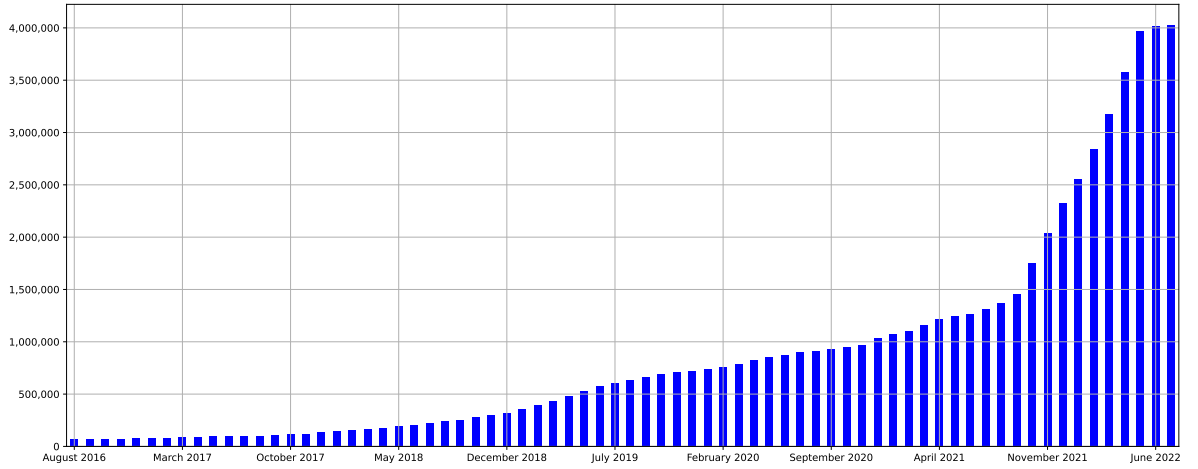


Figure 1.2: Growth of IoT/Linux malware in recent years.

Initially, early Mirai variants used only a set of usernames and passwords from a pre-configured list of 62 credentials to establish a connection via the Teletype Network (Telnet) protocol and gain access to the device. Nowadays, IoT malware is more sophisticated and incorporates a wide variety of exploits that affect different protocols and services running on device firmware. These exploits are publicly available on the Internet, and enable malware to gain access to devices and spread across the Internet. Commonly, IoT devices provide interfaces for device configuration and maintenance. This is because IoT devices typically do not have graphical interfaces as desktop systems do (e.g., mouse, video, etc.). Therefore, these services are exposed on the internet and present major security problems. Prior work such as [8], in which the firmware of different devices is analyzed on a large scale, discovered around 225 high-impact vulnerabilities in the web services that the devices exposed together with new ones in which backdoors were discovered in the firmware of the device for accessing it as an administrator, as is the case of the telephone and Private Branch Exchange (PBX) devices analyzed in [9].

This highlights the security problem in IoT system networks, in which keeping firmware/software updated is a major challenge and cybercriminals take advantage of known vulnerabilities to scan the network and carry out attacks to take control of devices. Therefore, it is important to analyze and understand threats in order to create tools and

## 1.1. Motivation and Justification

---

methods to assist security analysts as well as defenses to mitigate these types of attacks. One of the main techniques to observe new threats and attacks is the use of honeypots. This concept was introduced about two decades ago when the first Internet worms began to spread across Windows and Linux systems [10]. Basically, honeypots are vulnerable systems exposed on the Internet that are monitored and whose main purpose is to receive attacks in order to learn what actions are carried out by attackers and to create defenses against them in real systems [11]. Other works have focused on capturing attack vectors through honeypot systems. While [12, 13] analyzed attacks focused on the Telnet protocol, the authors of [14, 15] focused on protocols and services. On the one hand, the authors of [14] proposed a honeypot for the MQTT and Extensible Messaging and Presence Protocol (XMPP), emulating the device via Representational State Transfer (REST) Application Programming Interfaces (API). On the other hand, the authors of [15] proposed a system that scans real IoT devices by logging their responses and, through a reinforcement learning system, gives the attackers the best possible response. The authors of [16] introduced the idea of using Virtual Private Network (VPN) tunnels with the aim of having a real device listening in different geographical locations and simulating the existence of several devices. One of the main uses given to honeypots is that of capturing new families of malware that are trying to spread through the network, as well as the discovery of vulnerabilities that are being exploited in the wild [13, 17].

This highlights the importance of creating automatic tools for the analysis of malicious files, since manual analysis with both static and dynamic techniques is quite ineffective, requiring substantial effort and time on the part of an analyst. Unlike traditional systems, in which the x86 architecture predominates, IoT devices have a wide variety of architectures (e.g., MIPSel, ARMel, PowerPC, etc.). Due to this heterogeneity, the creation of runtime environments for the execution and analysis of samples is quite challenging since the malware could be intended for a specific device and not work on another. Currently, there are some frameworks for malware analysis on Linux/IoT-based systems [13, 18, 19, 20, 21, 22, 23, 24]. However, they only provide an analysis report and do not allow the correlation of the malware samples analyzed.

In the recent literature, the security of IoT has attracted the attention of the research community, with proposals based on assessing the current state of IoT (in)security from different points of view. At the firmware image level, the authors in [8, 25] perform a large-scale analysis of device firmware images, statically in [25] and dynamically through emulation in [8]. Firmware image emulation is quite challenging due to the heterogeneity of devices, and so the authors of [26] released a framework for firmware image emulation, in order to facilitate the large-scale security testing of firmware. The authors of [27] improved upon the framework in [26] by analyzing failure cases and creating different heuristics to improve the emulation rate of newer device firmware images.

In [28, 29, 30, 31] the authors conducted a large-scale analysis of IoT devices. In [28] the authors presented a systematic review of the state of the art of security in IoT, focusing on ever-evolving IoT vulnerabilities and performed the first large-scale analysis of IoT vul-

nerability exploits. In [29] a large-scale study of more than 1.3 million devices exposed on the Internet was carried out, and more than 25% of the devices analyzed had some known vulnerability. Unsurprisingly, around 2,600 devices could have been exploited with one of these vulnerabilities and become part of a botnet. In [30], the authors perform an analysis of the devices that exist in the internal networks of more than 16 million households. Currently, more than half of all houses have at least one IoT device. These devices have insecure protocols such as Telnet and File Transfer Protocol (FTP), and known vulnerabilities. Unlike the above works, [31] evaluated the insecurity of embedded devices, verifying the problem of the use of default passwords on a large scale throughout the world. The authors found that more than 500,000 devices, ranging from firewalls to VoIP adapters to printers, use default credentials. The authors of [32, 33, 34] studied the application layer of IoT platforms, and they discovered that there are design flaws and that most applications have more privileges than they really need. The authors of [35] focused on searching for vulnerabilities in the network interfaces of IoT devices using black box fuzzing.

Other studies have focused on malware that targets IoT devices. In [36], the authors conduct a study of the state of the art of IoT malware, classifying DDoS attacks on the basis of different characteristics (e.g., botnet architecture, scanning strategy, protocols). Authors in [18], perform a comprehensive analysis of malware targeting Linux-based IoT systems, both statically and dynamically, and present the main statistics regarding persistence, packing, anomalies in the Extensible Linkable Format (ELF) file format and the commands executed on the system. On the other hand, the authors of [19] carry out a study of malware in the IoT and the families that exist, summarizing the estimated size of the botnets as well as the timeline of the appearance of the family and the vulnerabilities used by them.

Other works in the literature, such as [6, 37, 38, 39, 40, 41], focus on specific malware families. In [37], the authors present an analysis of the malware known as Chuck Norris, while [39] performs a technical analysis, both statically and dynamically, of the Dofloo malware. In [38], the authors focus on two main methods of malware infection, namely brute force and the N-day vulnerability exploitation, analyzing Mirai as an example of the former and Bashlite and Darlloz as an example of the latter. In [6], the authors presented an exhaustive analysis of Mirai and its evolution in the first seven months of its life, demonstrating that malware that does not use sophisticated techniques is capable of compromising a large number of devices and threatening well-defended targets. In [40, 41], the authors presented an analysis of the Hajime botnet. The former analyzes aspects ranging from the initial stages of infection to the format of files used to store the configuration and payloads, while the latter analyzes the infected devices, the vulnerability that Hajime exploits, and the size of the botnet.

Other approaches are focused on creating models for malware classification, as in [42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52]. The works mainly differ in the size of the dataset and the algorithms they use. In [42, 43, 44, 45, 46, 47, 48] the authors use static features to generate the classification models, while in [49, 50, 51, 52] they focus on the use of dynamic features. Most of these works focus on binary classifications, that is to say they detect whether the

## 1.2. Objectives

---

sample is malware or not, although in some cases the models allow the classification of it into the best-known families of malware in the IoT (e.g., Mirai, Gafgyt, etc.).

In conclusion, security in the IoT ecosystem raises great concerns in the research world. The use of insecure protocols and default passwords continue to be two of the main problems, along with the use of outdated device firmware which is no longer maintained by the device manufacturer, leaving it exposed to known N-day vulnerabilities that are being exploited in the wild. Although machine learning algorithms for differentiating between malicious and benign files can be effective, their use on user devices may not be feasible in practice due to the limitations of capacity and memory that the devices have.

## 1.2 Objectives

In view of the problems described in Section 1.1, the main objective of this dissertation is the modeling and analysis of threats in the IoT environment, and the formulation of ways to combat them. In order to successfully achieve this objective, the following partial goals have been defined:

- **Goal 1.** The review of the proposals of other researchers regarding the classification and detection of threats, and security improvements in IoT environments. This objective is transversal throughout the doctoral thesis and will be considered in the entire research process. It is important to analyze the work of other researchers to see how they have solved the problems they are trying to address. This allows the learning of new methods, techniques and technologies, which enriches the research process.
- **Goal 2.** The analysis of attack trends that exist in the IoT ecosystem. This objective points to the creation of an observatory for the collection and modeling of existing threats in the IoT, allowing us to learn the behavior of attackers and the type of actions they perform on compromised systems. This allows us in turn to extract knowledge concerning the main attack tendencies that exist in the IoT and the Tactics, Techniques and Procedures (TTP) employed by the attackers.
- **Goal 3.** The design, implementation and evaluation of an architecture that allows the analysis of malicious files while addressing the heterogeneity of IoT device architectures. For several decades, malware was primarily designed to attack desktop computers. With the growth of what is known as the IoT, the number of malicious samples attacking these devices is growing exponentially. Due to the variety of architectures, this goal aims to build an analysis platform that supports the execution of malicious samples on different hardware architectures.
- **Goal 4.** The development of tools and techniques for assisting malware analysts and automating the process of relating malware samples to known threats and vetting unknown malware. To help malware analysts understand how harmful a new mal-



ware sample can be, this goal focuses on detailing the categorization and behavior of samples. The similarity between new samples and known threats will be the key to assessing the damage that new attacks can cause on the target system, and reducing their impact.

- **Goal 5.** The design, implementation and evaluation of a methodology for adding additional security measures to legacy IoT devices. IoT devices have known firmware vulnerabilities that are often trivial to exploit. Updating this firmware usually requires manual intervention and updates are not always available as the firmware is in many cases no longer maintained by the manufacturer. This objective points to the creation of a methodology for providing these firmwares with additional security measures in order to protect them against known threats (i.e., N-day vulnerabilities).

Figure 1.3 summarizes the main objective of this dissertation, describing an end-to-end solution that addresses: at 1) the collection and modeling of existing IoT threats, and a framework for 2) extracting features from collected samples, and 3) analyzing and characterizing them. Finally, when the collection of threats fails (e.g., unseen malware), this thesis then designs a framework to harden IoT devices, minimizing their attack surface.

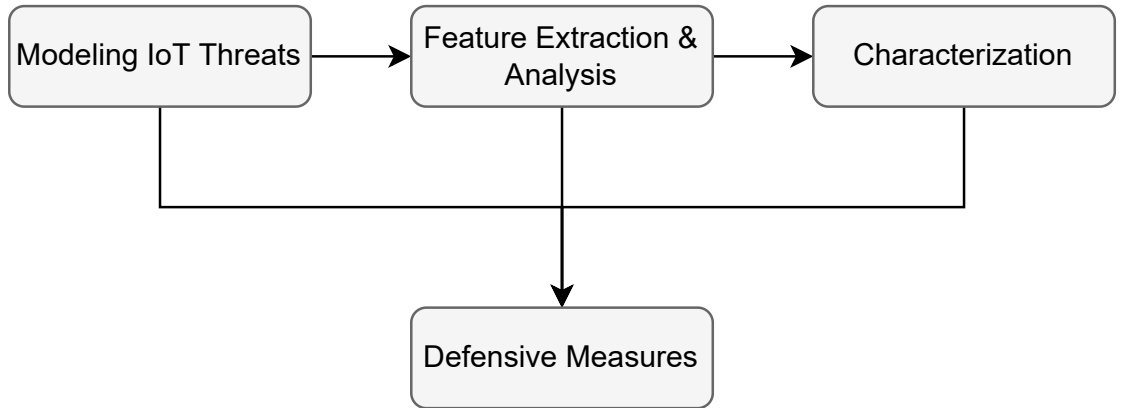


Figure 1.3: Scheme that describes the main objective of this doctoral dissertation.

### 1.3 Methodology and Work Plan

This section describes the methodology used to achieve each of the goals defined in Section 1.2, in order to fulfill the main objective of this doctoral thesis. The tasks carried out for each goal are the following:

- **Goal 1** – A review of the proposals of other researchers regarding the classification and detection of threats and security improvements in IoT environments.

The main purpose of this objective is mainly fulfilled in the initial stages of the doctoral thesis, allowing the researcher to become familiar with the field of threats, attacks and defenses in the IoT environment. However, once the objective is achieved, a periodic review of the state of the art will be carried out throughout the entire

### 1.3. Methodology and Work Plan

---

doctoral thesis since new research, approaches and technologies are continuously being developed by the scientific community. In this way, updating knowledge about emerging technologies is necessary in order to focus the research efforts on those aspects that the community still needs to solve.

- **Goal 2** — The analysis of attack trends that exist in the IoT ecosystem.

This objective refers to the study of attacks that occur every day in the IoT environment. First, an experiment is established to detect attacks against, and threats to, IoT devices through the Telnet and Secure Shell (SSH) protocol. Second, an experiment is performed to analyze what happens when attackers have access to valid VoIP service credentials.

The tasks necessary to achieve the first part of this objective are:

1. Establish a controlled environment and harden the server to prevent unauthorized access to services that are not the subject of this study.
2. Select a honeypot architecture that satisfies the characteristics of the experiment.
3. Create and configure virtual machines with an IoT architecture (i.e., ARM) to which attackers will have access.
4. Deploy a honeypot that will act as a proxy, redirecting traffic to the Telnet and SSH services of the virtual machines, and monitoring the connections and the actions of the attackers.
5. Analyze the information collected about the attacks received by the honeypot (i.e., analysis of the metadata of the connections, analysis of the interaction and analysis of downloaded files).

In order to achieve the second part of this objective, the steps to follow are:

1. Establish a controlled environment and harden those services that are not the subject of this study to prevent unauthorized access.
2. Define the account creation strategy using both weak and strong credentials.
3. Establish containment measures to avoid disturbing real phone users as well as request approval from the ethics committee to be able to carry out the experiments.
4. Deploy a PBX system and subscribe the accounts to dialplans that only allow calls to a set of countries, simulating a real system. Finally, these calls will be redirected to internal extensions of the system to avoid disturbing other users.
5. Leak accounts with strong credentials on paste sites and underground forums, in order to encourage fraudulent use of the system.

6. Analyze the data collected during the experiment (i.e., phone numbers, calls and fraud, Internet Protocol (IP) addresses).
- **Goal 3** — The design, implementation and evaluation of an architecture that allows the analysis of malicious files while addressing the heterogeneity of IoT device architectures.

Since the malware that targets IoT devices is compatible with several architectures due to the heterogeneity of IoT environments, it is necessary to have tools compatible with these architectures in order to analyze them. Therefore, an architecture for analyzing these malicious files can be designed by performing the tasks below:

1. Design an architecture for analyzing malware samples for the main architectures targeted by IoT malware, allowing the integration of different modules in order to cover the main phases in malware analysis.
  2. Prepare virtual machines for each architecture, which will be used as the base system and restored once the execution time of the malware has finished.
  3. Design a static analysis module to obtain information about the malicious file without executing it.
  4. Design the dynamic analysis module, which is responsible for executing the malware sample on its respective virtual machine and obtaining information about its execution.
  5. Design a module for the correlation of malware samples based on their static or dynamic characteristics.
  6. Evaluate the proposed architecture with a set of malicious samples designed to attack IoT devices.
- **Goal 4** — The development of tools and techniques for assisting malware analysts and automating the process of relating malware samples to known threats and vetting unknown malware.

The rise in the number of IoT devices together with the poor security that many of these devices present, either due to the use of vulnerable software and lack of updates or the misconfiguration of the devices by users, has caused the number of malicious samples that target these devices to grow significantly. This goal aims to design tools and techniques for assisting analysts in the process of matching malware samples. The tasks required to accomplish this goal are:

1. Design a methodology for modeling, discovering and vetting malware that is specially designed for Linux systems and IoT devices.
2. Perform data cleaning and preprocessing as well as an analysis and extraction of the main features.

## 1.4. General Discussion and Description of the Proposals

---

3. Implement a machine learning system for the classification of malware samples (i.e., Modeling phase).
  4. Characterize the samples that do not have labels in the dataset (i.e., Discovery phase).
  5. Cluster the unlabeled samples in the dataset to find the relationships between unknown and known samples (i.e., Vetting phase).
- **Goal 5** – The design, implementation and evaluation of a methodology for adding additional security measures to legacy IoT devices.

Due to the increase in the number of IoT devices and the absence of regular updates, this objective aims to create a methodology for providing devices with additional security measures. For the design, implementation and evaluation of the methodology, the following tasks should be performed:

1. Design and implement of a generic methodology for introducing additional measures in the device firmware and for protecting the main exposed services (i.e., HyperText Transfer Protocol (HTTP), Telnet, and UPnP).
2. Evaluate the effectiveness of the methodology by testing it with a set of firmware images from different vendors and devices. This task can be broken down into the following subtasks:
  - Collect of a dataset made up of real-world firmware from different devices and vendors.
  - Retrofit the firmware images with a Web Application Firewall (WAF) and a Secure Socket Layer (SSL) proxy to secure HTTP connections.
  - Evaluate the performance in terms of memory, Central Processing Unit (CPU) and the response time of the web application with respect to the firmware image without retrofitting through firmware emulation.
  - Test the methodology in a set of case studies with real devices.

## 1.4 General Discussion and Description of the Proposals

As mentioned above, the objective of this doctoral thesis is to analyze the attack trends in the IoT ecosystem and propose tools for facilitating the analysis of threats. In this section we provide a summary of the work carried out throughout the doctoral thesis. First, Section 1.4.1 gives a description of the proposals made to measure attacks within the IoT environment. Section 1.4.2 describes the tools for facilitating the analysis of malicious files that target IoT architectures. Finally, Section 1.4.3 describes a methodology for adding additional security measures to the firmware images of IoT devices.

### 1.4.1 Measurement of attack trends

This section describes the proposals related to the measurement of attacks related to the IoT environment, as well as the methodology implemented for the analysis of the collected data. Two honeypots were established in order to collect and model attacks in IoT environments for the Telnet, SSH and VoIP protocols.

On the one hand, we deployed the *Cowrie* [53] honeypot to monitor the attacks carried out against the Telnet and SSH services. We configured *Cowrie* to work in proxy mode, and its function was to redirect the traffic that these services received towards virtual machines compatible with Quick EMUlator (QEMU) [54] and that were IoT friendly (i.e., ARM). Thus, *Cowrie* becomes a high-interaction honeypot since the attackers deal directly with a complete system rather than an emulation of a file system and certain commands. For user authentication, we used the top 10 most common passwords used by malware that attacks IoT services as valid credentials [55]. Once an attacker successfully logs in, the traffic is redirected to a virtual machine. The honeypot was active for a period of 35 days, during which time the actions that were carried out in the system were monitored and the data corresponding to these actions were collected. Then, to evaluate the actions performed by the attackers, three types of analysis were carried out.

In the first place, we carried out an analysis of the metadata associated with the connections made to the honeypot, that is, the number of attacks received each day, the source IP addresses of the connection, the connections that are attempted through SSH port forwarding and the most frequently used users and passwords.

The second analysis consisted of a classification of the attackers' interaction with the honeypot. To accomplish this task, we extracted all the command sessions of the attackers, and then we performed a cleanup and standardization process. Thus, we eliminated duplicate sessions (i.e., those that belong to the same IP and in which the commands entered are exactly the same), and standardized the commands by removing or replacing specific semantics that may be variable but which correspond to the same command (i.e., IP addresses, Uniform Resource Locators (URLs), download filenames, payloads, SSH keys and similar information). Then, we separated each of the commands entered, generating an ordered list in which each element is a command. Finally, given the sequences  $A$  and  $B$ , we transformed each of them into a vector,  $\vec{A} = \{a_1, a_2, \dots, a_n\}$  and  $\vec{B} = \{b_1, b_2, \dots, b_n\}$ , where  $a_i$  and  $b_i$  represent the number of times that the command in position  $i$  appears in the sequence, and  $n$  is the total number of different commands when comparing the two sequences. To measure the similarity between two vectors, we used the cosine similarity, which is computed as follows:

$$\text{similarity}(A, B) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \times \|\vec{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}} \quad (1.1)$$

## 1.4. General Discussion and Description of the Proposals

The result of the cosine similarity belongs to the interval  $[0,1]$ , with 1 being a perfect similarity and 0 indicating there is no similarity between the two lists of commands.

Finally, the last analysis corresponded to the study of the correlation between the files downloaded by attackers in the honeypot. For this purpose, we executed the downloaded binary files and stored the trace of system calls made by these files. To calculate the similarity between two samples, it is necessary to extract the sequence of calls of size  $N$ , known as an  $N$ -gram. Once the  $N$ -grams had been extracted, we used the Jaccard index [56], as given by the following expression, to obtain the similarity between the two sets:

$$Jaccard\_index(s_1, s_2) = \frac{|s_1| \cap |s_2|}{|s_1| \cup |s_2|}, \quad (1.2)$$

In this expression, the numerator represents the number of subsets that appear in both samples, and the denominator represents the total number of unique subsets in the two both samples. The similarity between sets obtained as a result belongs to the interval  $[0,1]$ .

On the other hand, we set up a custom VoIP honeypot based on Asterisk, an open-source PBX, to measure what type of actions are preformed when valid credentials are obtained for VoIP services, which are increasingly used by users and devices. In this regard, this honeypot monitored the accesses and actions through it to understand how these services are misused. In order to achieve this, we designed two experiments for this honeypot. The first one employed user accounts with weak credentials that were easily guessable through brute force processes, while the second one had user accounts with strong credentials that were leaked on pastebin sites and underground forums. This allowed us to study whether the actions taken by attackers differ depending on how they gained access. In order to make the system more realistic, these user accounts were linked to different contexts, only allowing calls to destinations in the United States, the United Kingdom and Spain. It should be noted that when the call was directed to these destinations, it was redirected to internal extensions to make the attackers believe that the call was being made correctly. Figure 1.4 shows the methodology followed in this proposal.

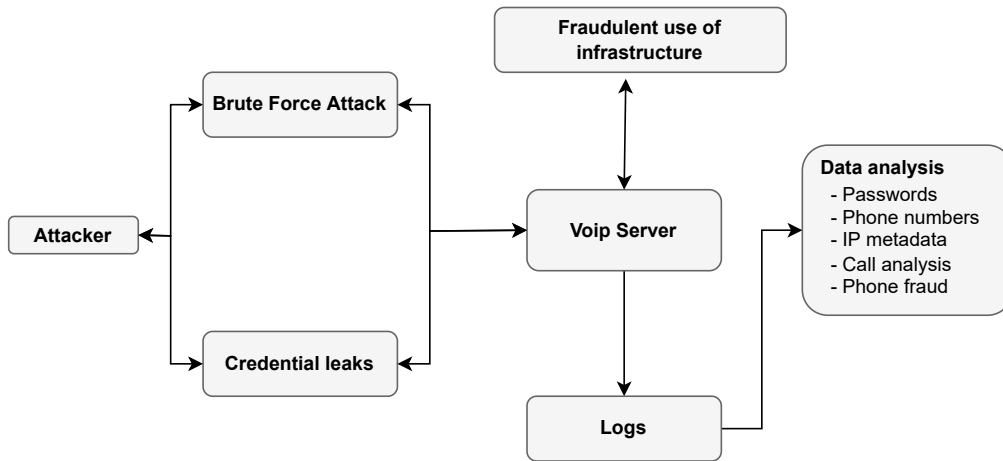


Figure 1.4: Methodology followed to evaluate the actions carried out in the VoIP honeypot.

Once the two experiments were completed, we performed a data-driven analysis of the actions that were carried out by the attackers to understand how these services are used by attackers. In the first place, we performed an analysis of the source IP addresses in order to establish the origin of the attacks, whether they belonged to blacklists of known proxies or bots, and the Autonomous System Name to which they belonged. Secondly, we analyzed the passwords used by the attackers to try to access the system. Since in current versions of the Session Initiation Protocol (SIP), the client ends up computing the hash of the username and password in the authentication process, as described in Algorithm 1.3, the server never sees the password in plain text. Therefore, we carried out a brute force attack with a dictionary of the most commonly used passwords to find the most frequently used passwords in our system.

$$\begin{aligned}
 A &= MD5(user : realm : password) \\
 B &= MD5(method : sip : uri) \\
 hash &= MD5(A : nonce : B)
 \end{aligned}
 \tag{1.3}$$

Then, we analyzed the phone numbers that the attackers tried to call, extracting the following information:

- Phone number called, that is, whether it was a valid number or not based on the patterns that phone numbers have throughout the world.
- Localization of the phone number based on the country and area code.
- Type of telephone number, that is, whether it belonged to a mobile, land-line, special rate number, etc.

Finally, we evaluated the calls made by the attackers, checking their timestamp to analyze whether they took into account the local time of the destination of the call. We also analyzed the type of fraud that was attempted. For this purpose, we applied the taxonomy described in [57], which systematically explores the different types of fraud existing in telephone networks. Figure 1.5 describes how we applied the taxonomy in our scenario to identify the type of fraud being attempted through our system.

As part of this investigation, whose results are briefly summarized in Section 1.5 and detailed in Chapter 3, through the honeypot we were able to identify and characterize the different types of calls that were made. Due to the increase in frequency of this type of attack, together with the low prices that call plans present in some countries, these systems are exploited in spam and scam campaigns. Countermeasures such as blocking IP or phone numbers are not enough since criminals can change their IP address (e.g., via The Onion Router (TOR)) or the phone number used to call (e.g., via Caller-ID Spoofing). In this regard, we propose a system for automatically answering phone calls which is responsible for converting the incoming audio to text and feeding a chatbot that is responsible for giving an appropriate response. Then, this response is converted to audio format in real time and played in the context of the conversation in order to deplete the scammer's resources,

## 1.4. General Discussion and Description of the Proposals

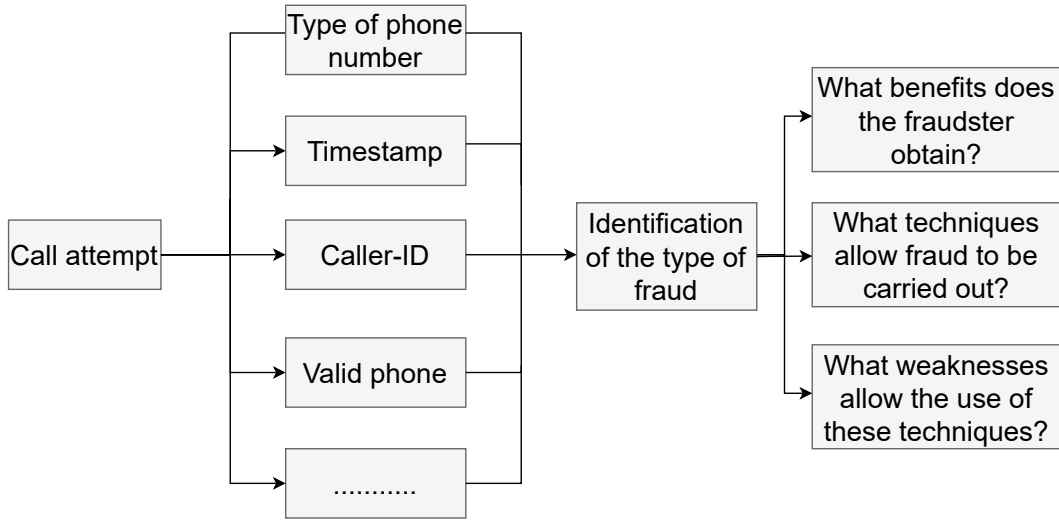


Figure 1.5: Methodology that has been followed to identify the type of fraud.

as the time spent talking to a chatbot is not used to deceive other victims. In total we built two chatbots: a rules-based one built on top of Rasa Natural Language Understanding (NLU) [58], and the other using OpenAI’s Generative Pre-trained Transformer 3 (GPT-3) model [59]. Finally, the call feeds a classifier that categorizes the type of call that has been received. Figure 1.6 shows the scheme of the system architecture.

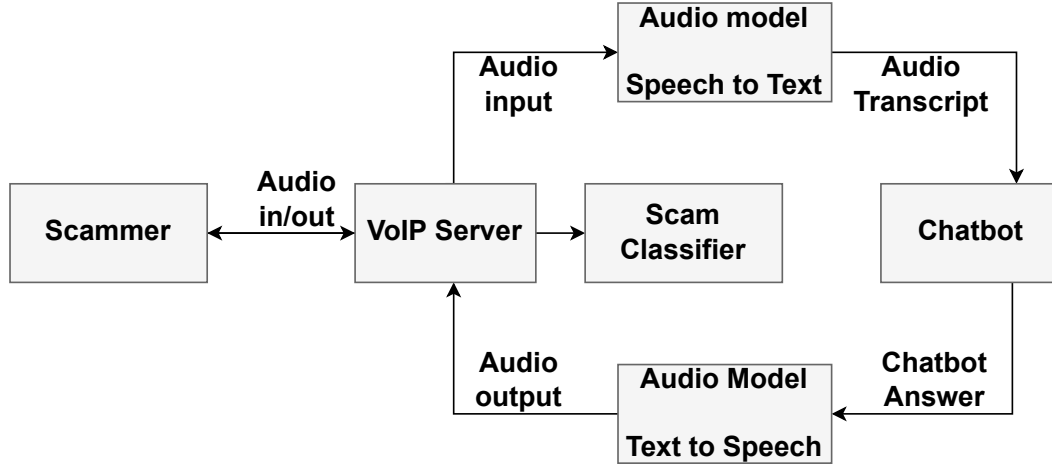


Figure 1.6: Scheme of the system architecture.

### 1.4.2 Tools and techniques to assist malware analyst

This section describes the framework for analyzing malware samples targeting IoT devices, as well as the methodology for modeling, discovering, and examining malware samples targeting Linux-based IoT devices.

IoT devices often contain very weak or not security measures, and the malware that targets these devices increases every day. Unlike classical malware that targets desktop



systems, in the IoT environment there are multiple hardware architectures (e.g., ARM or MIPS), which makes analysis difficult since it is necessary to have these architectures, either physically or emulated, to dynamically analyze malware samples. In this regard, we propose a modular solution to analyze malware samples for different hardware architectures (i.e., ARM, MIPSel, MIPSb, PowerPC, 80386, and x86-64). Figure 1.7 shows a scheme of the architecture and the modules that make it.

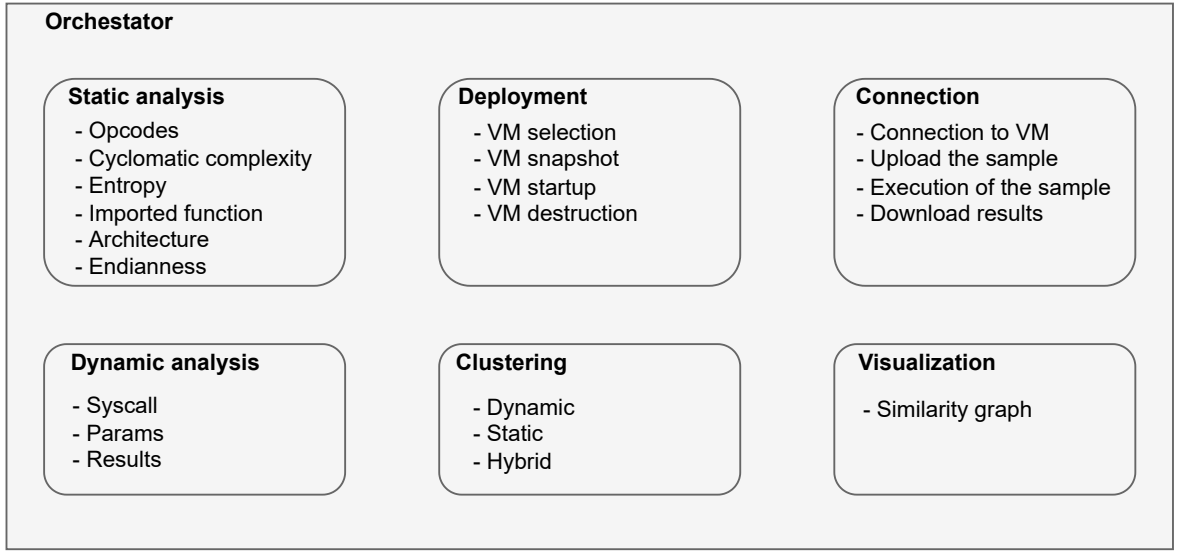


Figure 1.7: Block diagram of the architecture designed.

As can be seen in the diagram, the architecture is made up of six modules, which are managed by the orchestrator. Thanks to its modular structure, each module can be used independently. The description of each module is as follows:

- **Static analysis:** This module is built on top of *radare2* [60] and is responsible for parsing ELF files and collecting executable information such as entropy, the cyclomatic complexity of each function, the sequence of disassembled code opcodes, sections, imported functions, text strings, the architecture, endianness, etc.
- **Deployment:** This module is responsible for starting the virtual machine, shutting it down, or restarting it. It is necessary to indicate the architecture and the endianness of the architecture in order to manage which virtual machine it is necessary to start.
- **Connection:** This is responsible for establishing the connection with a remote host through the SSH service as well as sending commands. It also allows uploading and downloading files from a remote host. In this case, the orchestrator sends the sample to be analyzed, and information about its behavior is extracted with *strace*. The file with the execution trace is downloaded before shutting down and restoring the machine to its original state.
- **Dynamic analysis:** This preprocesses and extracts the system calls as well as the parameters used and the results of an execution trace.

## 1.4. General Discussion and Description of the Proposals

---

- **Clustering samples:** This is responsible for grouping binary files based on previously extracted features. To calculate the similarity between two samples, we follow three approaches, depending on the characteristics used:

1. **Dynamic approach:** This uses the execution traces obtained dynamically, generating sequences of size N, known as N-grams. Finally, their similarity is calculated using the Jaccard index, as presented above in Formula 1.2.
2. **Static approach:** We use two metrics to measure similarity statically. The first one uses sequences of opcodes of size N extracted from the disassembled code, and the similarity is calculated in a similar way to that used in the dynamic approach but with opcodes instead of system calls. The second one uses the cyclomatic complexity of each of the functions that exist in the binary. Given two executables, we represent the cyclomatic complexity of each of them as a vector where each position represents how many functions have the cyclomatic complexity represented by that position. We normalize both vectors to contain the same number of elements, and finally, we use a distance function to calculate the similarity between them, which is formalized as follows:

$$distance(s_1, s_2) = \sum_{i=0}^{|F|} \frac{\min(f_i^{s_1}, f_i^{s_2})}{\max(f_i^{s_1}, f_i^{s_2})} \times \frac{1}{|F|} \quad (1.4)$$

3. **Hybrid approach:** This allows the use of all the similarity indices described above, or a subset of them, assigning them a weight as a percentage. The final index is the sum of all the recalculated similarity indices.
- **Visualization:** This generates a graph in which the nodes represent the malware samples and the edges indicate similarity if there is a degree of similarity greater than an established threshold.

The orchestrator is in charge of interconnecting all the modules. First, it uses the static analysis module to obtain the binary information that will be used for the next phase. Once the architecture is detected, it is responsible for deploying the virtual machine and uploading the malicious file to the system. Then, by giving it execution permissions, it can run for a predetermined amount of time. Once the execution time has elapsed, it downloads the execution traces and recovers the original state of the virtual machine. Finally, the similarity with other previously analyzed samples is calculated.

As mentioned above, malware attacks on IoT devices are constantly increasing due to the lack of proper configuration and outdated software on IoT devices. This causes new malware variants to appear every day, and it is important to understand their main characteristics and be able to relate new threats to known samples. With this in mind, we propose a methodology for characterizing malware samples and understanding their main features.

Figure 1.8 shows the scheme of the proposed methodology, which is composed of the following three phases:

- **Modeling:** The first phase deals with feature extraction and serves as an exploration to guide the feature engineering process. These features feed a supervised machine learning model for characterizing known threats.
- **Discovery:** This phase aims to characterize a set of samples that are not labeled using the modeling phase classifier. Since there is no label, it is not possible to quantitatively ensure that the classifier's predictions are correct. Therefore, we compute the level of similarity between the labeled and unlabeled samples at the sequence level of opcodes of size  $N$  (N-grams) to generate an initial ground-truth. This allows us to check whether the family predicted by the classifier has a high percentage of similarity at the opcode level with other labeled samples of that family. It should be noted that the classifier was formally evaluated in the previous stage using k-fold cross-validation and that this phase aims to discover new threats.
- **Vetting:** This phase is responsible for characterizing new threats among the unlabeled samples. To do this, an unsupervised clustering process is performed, driven by a custom distance function that falls back on the features extracted in the modeling phase and the similarity index of the discovery phase. The distance function is formalized as follows:

$$distance(s_1, s_2) = \frac{\sum_{i=0}^{|F|} \frac{\min(f_i^{s_1}, f_i^{s_2})}{\max(f_i^{s_1}, f_i^{s_2})} \times \frac{1}{|F|} + jacc(s_1, s_2)}{2} \quad (1.5)$$

In this expression,  $f_i^{s_1}$  is the feature  $i$  of sample  $s_1$  (same for  $s_2$ ) and  $|F|$  is the total number of features. For the clustering of samples, those samples with an index greater than 0.8 have been considered to be similar, that is, they have a similarity ratio of 80%. The choice of this similarity percentage is detailed in Chapter 6.

### 1.4.3 Defensive measures to protect IoT devices

This section describes the proposed methodology for retrofitting additional security measures into the firmware of legacy IoT devices to make them more secure from known attacks.

Due to the extensive network of obsolete IoT devices, this methodology focuses on those devices that have legacy or End-Of-Life (EOL) firmware. This is mainly because devices containing legacy firmware tend to be vulnerable, and vulnerable services are likely to be directly exposed to the Internet. In addition, legacy firmware will not receive any security patches from vendors. Therefore, a system for HArdening LEgacy Internet-of-Things (HALE-IoT) devices with legacy firmware is proposed. Figure 1.9 depicts a high-level view of a hardened device with additional security measures. In the first place, we can observe the vulnerable services, which are launched when the device starts (e.g., HTTP web server,

## 1.4. General Discussion and Description of the Proposals

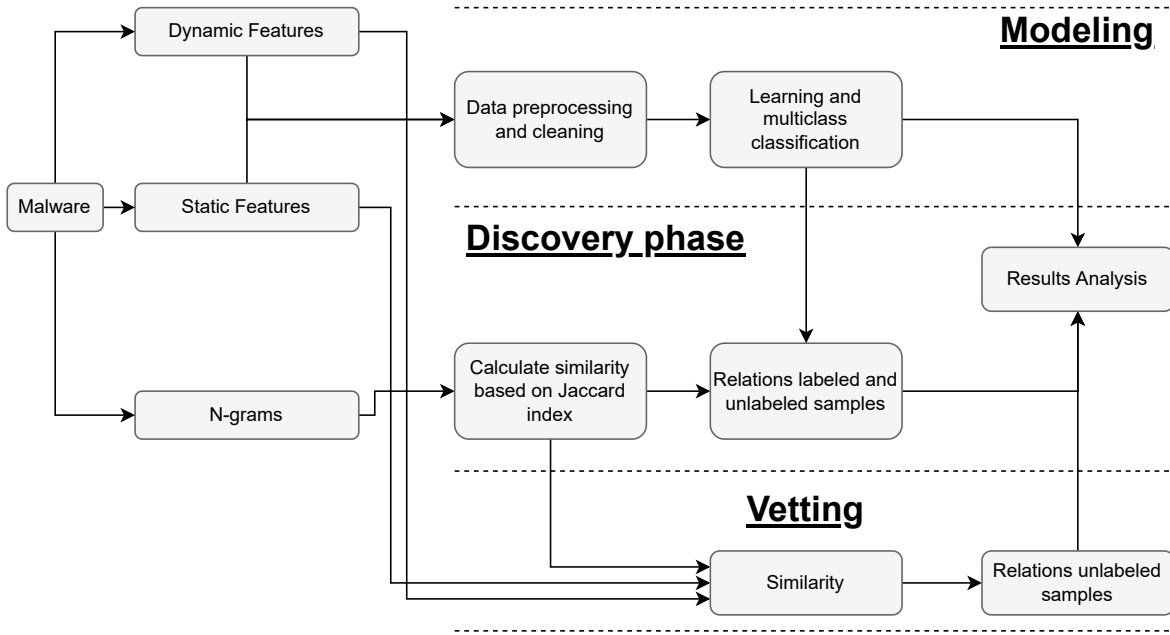


Figure 1.8: Block diagram of the malware sample characterization methodology.

FTP, Telnet, etc.) and, on the other hand, the file system where the binaries and configuration files of these potentially vulnerable services are stored. The entry point to these services can also be seen, usually through a network interface, representing the address 0.0.0.0 as an interface that would be accessible to an attacker.

For the design of the methodology, we took into account that the hardening process has to be flexible and capable of accommodating the main services used in IoT devices. The fail-safe minimization principle was also followed, whereby modifications made to the firmware should be as unintrusive as possible and should not interfere with the operation of the device. The methodology was designed to make as few changes to the device as possible and to harden the system through changes to configuration files. Modifications should only be made at the binary code level when it is not possible to reconfigure the device by other means. In this scenario, the process should be guided by a human to produce a proof of concept before reproducing the patch and retrofitting it to scala automatically.

In order to test the feasibility of the methodology in a practical scenario and address the challenge of retrofitting and hardening a set of IoT devices, we designed a system to satisfy the following points:

- **Secure front-end:** Its objective is to provide insecure services with a secure interface through the development of wrappers, guaranteeing confidentiality and secure access control. At this point, an SSL proxy will be retrofitted to provide a cryptographic update in case the device lacks it (e.g. HyperText Transfer Protocol Secure (HTTPS) instead of HTTP, SSH instead of Telnet, etc.), protection from SSL/Transport Layer Security (TLS) attacks, and patches for weak SSL/TLS configurations.

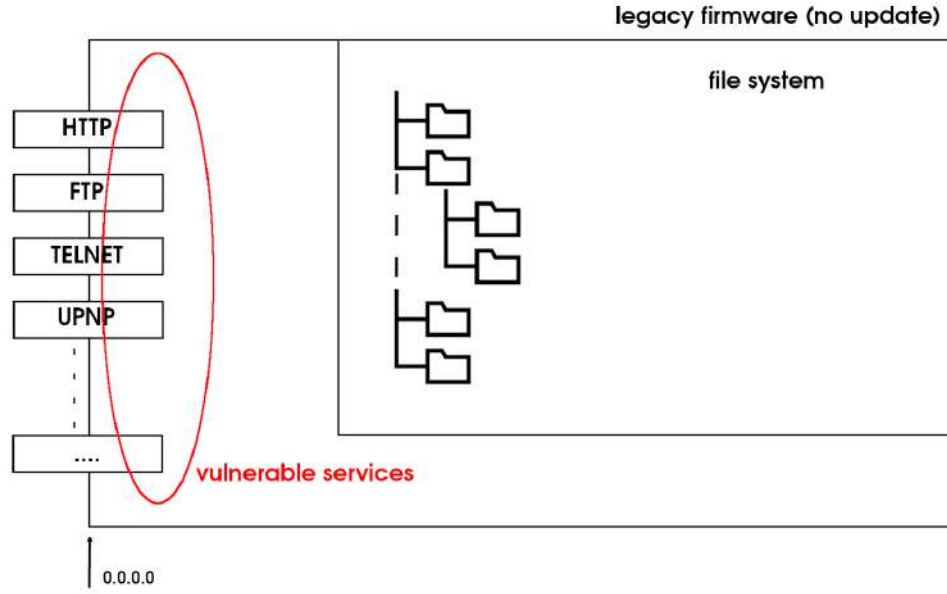


Figure 1.9: IoT firmware system diagram.

- **Proactive detection:** Its purpose is to offer protection at the application level through the retrofitting of a firewall in the specific domain of the application to be protected (e.g. a WAF for web requests).
- **Advanced-level access:** Its objective is to harden the device administration interface since IoT devices generally do not have a graphical user interface and are managed remotely.

Figure 1.10 shows a high-level diagram of the generic hardening of an IoT device. It can be seen that, unlike in Figure 1.9, the vulnerable services are not directly exposed to the internet but rather listen on the localhost interface. Secure interfaces are found through the retrofitted wrappers in the device firmware and the specific firewall in the application domain. Finally, the HALE-IoT admin panel can also be viewed to provide access for configuration updates, updates to the xAF rule set, and bug fixes.

## 1.5 Results

This section presents the results obtained for each of the goals described in Section 1.2. The results include a brief description of each publication together with the most relevant conclusions.

Regarding Goal 1, there is no specific publication included, since this objective involves the review of the state of the art, which has been applied throughout this thesis. The knowledge acquired by accomplishing this objective constitutes the foundation on which the rest of the goals of this thesis project are based, since it is important to know the state of the art

## 1.5. Results

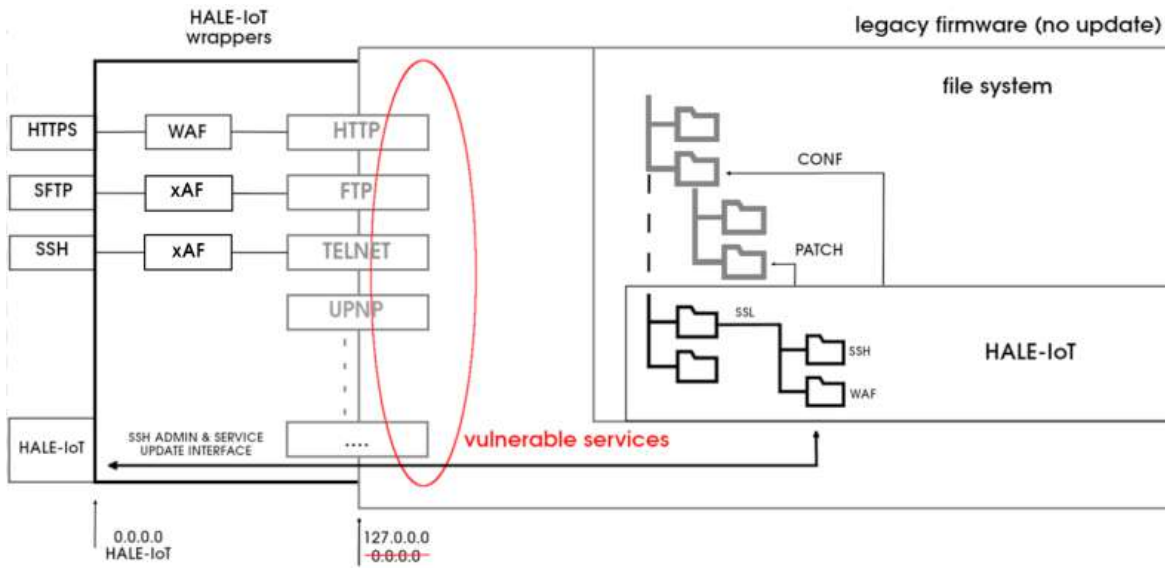


Figure 1.10: IoT firmware system diagram AFTER hardening.

and how other proposals in the literature address the different problems that the said goals try to solve.

In order to fulfill Goal 2, two honeypots were deployed to measure existing attacks on Telnet, SSH and VoIP protocols. In the first place, as specified in Section 1.4.1, a high interaction honeypot for Telnet and SSH services was deployed using the *Cowrie* honeypot in proxy mode, with the requests being directed to an emulated virtual machine with ARM architecture. In this work, we have performed a statistical analysis of the connections that were made to our honeypot. The results show that brute-forcing usernames and passwords remains one of the preferred methods for attackers to gain access to devices and spread their malware. Through the classification of the scripts introduced once they gained access, it can be seen that they are mainly attacks carried out automatically and that they use similar TTP to download and install malware on vulnerable devices since most of the command sessions are grouped into 7 or 8 clusters. The 710 unique files downloaded by the attackers were also analyzed, most of them being ELF binary files. Through the calculation of the similarity between each pair of collected samples, it can be observed that 70% of the samples are grouped in the same cluster and that they belong to the Hajime family, specifically to the downloader used in the first stage of the chain of infection. This research was submitted for publication in the *Computers & Electrical Engineering Journal* [61].

Second, as described in Section 1.4.1, a custom honeypot was deployed to understand how VoIP services are exploited when attackers obtain valid device credentials. To do this, two experiments were launched, one with weak credentials that were vulnerable to brute force attacks and the other with strong credentials, which were leaked in different paste sites and underground forums. The results show that attackers exploit VoIP systems to carry out different types of fraud in order to obtain economic benefits. Different types of fraud were identified, with Toll Evasion and Revenue Share being the most prevalent. Calls

made to personal phone numbers were also identified, which suggests that these attacks are also used for spam or scam campaigns. Through the study of the numbers that the attackers tried to call, calls to numbers with different prefixes were identified, indicating an attempt to avoid dialplan restrictions. Although some calls seemed random, others were focused on the destination they were trying to call in order to maximize the success of their attacks (i.e., calls were made during office hours or when people were at home). A correlation was made between the valid numbers to which calls were tried and the origin of the call, creating different clusters of telephone fraud campaigns. Finally, a series of countermeasures were proposed to avoid automatic attacks on this type of service. The results of this research work were published in the journal *Computers & Security* [62].

To conclude Goal 2, we proposed an architecture capable of functioning as a honeypot against criminals who use phone calls to carry out their attacks. First, we built a scam/spam call classifier using different machine learning models. The results show that the SVM model with the linear kernel is the one that provides the best results. Then, we built two chatbots using different strategies: the first one based on rules using Rasa NLU and pre-recorded audio, and the second one based on the GPT-3 model and audio generated in real time. The chatbots were evaluated with real calls corresponding to a specific type of scam known as Tech Support Scam. The chatbot built on Rasa NLU managed to maintain the call for around 4.2 minutes, and the chatbot built with the GPT-3 model did so for 3.8 minutes. The results of this study were submitted for publication in the journal *Institute of Electrical and Electronics Engineers (IEEE) Access* [63].

Regarding Goal 3, which corresponds to the design and implementation of a framework for the analysis of malicious files targeting IoT devices, the corresponding proposal, which was discussed in Section 1.4.2, was evaluated using different malware samples that were distributed over five hardware architectures. The samples were tagged using AVClass, which is a tool that categorizes the samples based on the tags provided by the different antivirus engines used by VirusTotal. We then extracted the static and dynamic characteristics of each of the samples in our dataset, and finally, we clustered the samples using different approaches: 1) opcode sequences, 2) cyclomatic complexity, and 3) system call sequences. The results show that these types of techniques are effective in clustering malware in the IoT, providing a way to analyze a set of samples and automatically correlate them based on their static or dynamic characteristics. Dynamic features allow us to relate samples from different architectures, while static features based on disassembled code depend on the architecture and its Instruction Set Architecture (ISA). Because of this, samples from different architectures cannot be related. In addition, static features can be affected by code obfuscation, by the compiler used, or by the compiler options used to build the sample. The results of this proposal were published in the journal *Security and Communication Networks* [64].

For the fulfillment of Goal 4, a methodology was developed to assist malware analysts in the classification, discovery and vetting of new threats, as described in Section 1.4.2. In this regard, we used the proposed methodology to systematize Linux-based malware analysis, showing that malware targeting architectures such as x86-64 is quite different from mal-

## 1.5. Results

---

were targeting IoT architectures. We also showed that the use of Machine Learning (ML) to understand large sample datasets is a challenge, due to the noise that appears in the dataset, a fact that is mainly influenced by the lack of consensus among antivirus engines and the use of publicly-available code from the Internet. This study also presents an in-depth analysis of previously unlabeled threats through case studies of the clusters with the largest population. By using reverse engineering techniques in these studies, some samples from each cluster were checked to verify whether the similarity was correct and to characterize its functionality. Among the unlabeled threats, different samples were found that contained encrypted bash files that would download malware for cryptomining campaigns, malware written in the Go language for spreading through SSH services, exploits, and shellcodes, among others. The results of this proposal were published in the journal *Future Generation Computer Systems* [65].

Finally, Goal 5 proposed a methodology for adding defensive measures to the firmware of legacy IoT devices, as described in Section 1.4.3. To evaluate the proposed methodology, the web services of 395 emulated images were protected. To do this, defensive implants were added to protect the application layer with a WAF and an SSL proxy to provide HTTPS support to devices. Through the emulation of the firmware images, performance measurements were made before and after retrofitting the images with the defensive implants, performing a set of common requests and another set, of similar size, of attacks for which there were rules in the WAF. Finally, the effectiveness was evaluated through proofs of concept with firmware vulnerable to known exploits, allowing them to be detected through rules and then seeing whether their exploitation was prevented. As a final part of the evaluation, HALE-IoT was tested in several case studies with real devices having a storage space that varied from 8 MB to 4096 MB, and with a Random Access Memory (RAM) memory from 32 MB to 1024 MB and different hardware architectures. The results of this proposal were submitted for publication in the *Internet of Things* journal [66].

## Summary of Results

All the goals proposed in Section 1.2 have been achieved by following the tasks defined in Section 1.3. All the results of the work carried out in this doctoral thesis are summarized in the list of publications below:

- **Goal 2** — Analysis of the attack trends that exist in the IoT ecosystem.
  - *Hajime's Return: Stories from a Customized Honeypot for the IoT*, submitted to *Journal of Internet Technology*. Journal paper. JCR2021 Q4 (148/164) in COMPUTER SCIENCE, INFORMATION SYSTEMS, IF 1.140 [61].
  - *On how VoIP attacks foster the malicious call ecosystem*, published in *Computers & Security*. Journal paper. JCR2021 Q2 (42/164) in COMPUTER SCIENCE, INFORMATION SYSTEMS, IF 5.105 [62].



- *A chatbot for selectively engaging with online fraudsters*, submitted to *IEEE Access*. Journal paper. JCR2021 Q2 (79/164) in COMPUTER SCIENCE, INFORMATION SYSTEMS, IF 3.476 [63].
- **Goal 3** – Design, implementation and evaluation of an architecture that allows the analysis of malicious files while addressing the heterogeneity of IoT devices.
  - *Automatic Analysis Architecture of IoT Malware Samples*, published in *Security and Communication Networks*. Journal paper. JCR2020 Q4 (125/161) in COMPUTER SCIENCE, INFORMATION SYSTEMS, IF 1.791 [64].
- **Goal 4** – Design of tools and techniques to assist malware analysts and automate the process of relating malware samples to known threats and vetting unknown malware.
  - *Characterizing Linux-based malware: Findings and recent trends*, published in *Future Generation Computer Systems*. Journal paper. JCR2020 Q1 (7/110) in COMPUTER SCIENCE, THEORY & METHODS, IF 7.187 [65].
- **Goal 5** – Design, implementation and evaluation of a methodology for adding additional security measures to legacy IoT devices.
  - *HALE-IoT: HARdening LEGacy Internet-of-Things devices via retrofitting defensive firmware modifications and implants*, submitted to *Internet of Things Journal*. Journal paper. JCR2021 Q1 (9/164) in COMPUTER SCIENCE, INFORMATION SYSTEMS, IF 10.238 [66].

## Other Results

These include other proposals resulting from collaboration with other researchers that are published. These proposals aim to solve certain cybersecurity problems but are not specifically within the scope of this doctoral thesis. These proposals are the following:

- Forensic analysis in the IoT.
  - \* *Non-Volatile Memory Forensic Analysis in Windows 10 IoT Core*, published in *Entropy*. Journal paper. JCR2019 Q2 (33/85) in PHYSICS, MULTIDISCIPLINARY, IF 2.494 [67].
  - \* *A context-centered methodology for IoT forensic investigations*, published in *International Journal of Information Security*. Journal paper. JCR2021 Q2 (52/110) in COMPUTER SCIENCE, SOFTWARE ENGINEERING, IF 2.427 [68].
  - \* *Developing an IoT forensic methodology. A concept proposal*, published in *Forensic Science International: Digital Investigation (continuation of the journal Digital Investigation)*. Journal paper. JCR2021 Q4 (131/164) in COMPUTER SCIENCE, INFORMATION SYSTEMS, IF 1.805 [69].

## 1.5. Results

---

- \* *Forensic Analysis of the Xiaomi Mi Smart Sensor Set*, published in *Forensic Science International: Digital Investigation (continuation of the journal Digital Investigation)*. Journal paper. JCR2021 Q4 (131/164) in COMPUTER SCIENCE, INFORMATION SYSTEMS, IF 1.805 [70].
- Vulnerability detection.
  - \* *An instrumentation based algorithm for stack overflow detection*, published in *Journal of Computer Virology and Hacking Techniques*. Journal paper. Not indexed [71].
- IoT pattern attack detection.
  - \* *Attack pattern recognition in the Internet of Things using complex event processing and machine learning*, presented at the *2021 IEEE International Conference on Systems, Man, and Cybernetics*. Conference paper. GGS Rating A- [72].

## CHAPTER 2

# Hajime's Return: Stories from a Customized Honeypot for the IoT

- **Title:** Hajime's Return: Stories from a Customized Honeypot for the IoT.
- **Authors:** J. Carrillo-Mondéjar, J.M. Castelo Gómez, J. Roldán-Gómez, Sergio Ruiz Villafranca and G. Suarez-Tangil.
- **Type:** Journal paper.
- **Journal:** Journal of Internet Technology.
- **Publisher:** Library & Information Center.
- **ISSN:** 1607-9264.
- **Status:** Under Review.
- **JCR IF/ranking:** 1.140/Q4 (JCR2021).

---

# Hajime's Return: Stories from a Customized Honeypot for the IoT

J. Carrillo-Mondejar<sup>1</sup>, J. Roldán-Gómez<sup>1</sup>, J.M. Castelo Gómez<sup>1</sup>, Sergio Ruiz Villafranca<sup>1</sup>, G. Suarez-Tangil<sup>2</sup>

<sup>1</sup>Albacete Research Institute of Informatics, University of Castilla-La Mancha, Spain

<sup>2</sup>IMDEA Networks Institute, Spain

{javier.carrillo, jose.roldan, juanmanuel.castelo, sergio.rvillafranca}@uclm, guillermo.suarez-tangil@imdea.org

## Abstract

Since the inception of the Internet of Things (IoT), the security measures implemented on its devices have been too weak to ensure the appropriate protection of the data that they handle, thus making this environment the perfect one for cybercriminals to carry out their attacks. In order to do so, they continuously seek out for vulnerable units so that they can control as many as possible, leading to attacks spreading through networks and infecting a high number of devices. At the same time that the IoT has evolved to provide a higher degree of security, the techniques used by attackers have done so as well. This has led to the need of continuously studying the way in which these attacks are performed in order to gather significant knowledge for the development of the pertinent security measures.

In view of this, this article analyses the state of IoT attacks by developing a high-interaction honeypot for Secure SHell (SSH) and Teletype Network (Telnet) services that simulates a custom device with the Advanced Reduced Instruction Set Computer Machine (ARM) architecture, which is the most commonly used in this environment. By doing so, we make it possible to analyse the attacks on these services, which have been on the rise in the recent years both in terms of quantity and effectiveness. This study is carried out in two steps. Firstly, we analyze and classify the interaction between the attacker and the devices by clustering the commands that they sent in the compromised Telnet and SSH sessions. Secondly, we study the malware samples that are downloaded and executed in each session and classify them based on the sequence of system calls that they execute at runtime. In addition, apart from studying the active data generated by the attacker, we extract the information that is left behind after a connection with the honeypot by inspecting the metadata associated with it.

**Keywords:** Honeypot, Malware, IoT, Data analytics, Expert systems

## 1 Introduction

Nowadays, there are innumerable devices connected to the Internet which interact with each other, providing services to users that, until a few years ago, seemed unthinkable. The result of integrating technology in new environments different from conventional ones, namely the cloud, desktop or mobile, is what we know as the Internet of Things (IoT). IoT devices make people's lives

easier by easing tasks that users perform on a daily basis. Using a mobile device for controlling household appliances, or utilizing your own voice for turning on the light or the television are some examples.

This means that the IoT is heavily involved in the activities that a person carries out in many aspects of their life. As a consequence, due to the high number of interactions that are made between user and device, both actively and passively, the resulting volume of data that is managed in this environment is immense. In addition, since some of this data can have a high degree of sensitivity, cybercriminals find it very appealing to attack IoT units. Unfortunately, although these devices provide a great number of features that are attractive to users, the security measures implemented on them are not strong enough to stop these attacks, thus making them vulnerable due to such simple aspects such as using default and easily guessable user and password combinations, having weak default settings or running well-known-to-be outdated and vulnerable software.

The characteristics mentioned above, together with the lack of knowledge that many users have regarding the use of new technologies, have led cybercriminals to focus many of their efforts on attacking IoT devices and obtaining financial returns from them. According to a recent report [1], the number of attacks on these devices reached one hundred million in 2019, exceeding the number of attacks in 2018 by seven times.

Under these circumstances, it is important to understand what activities attackers perform in order to compromise IoT devices and what actions they carry out once they have gained access to a system. To study this, the use of honeypots, which are devices which simulate systems that present some kind of vulnerability, is a very effective approach to attract attackers, and a technique that has been successfully used in other experiments such as [2]. This allows for different types of attacks to be captured and, upon analysis, the extraction of knowledge regarding the multiple techniques and tools used by attackers for carrying out their criminal schemes.

In this work, we have deployed a high interaction honeypot [3] that simulates a device with the Advanced Reduced Instruction Set Computer Machine (ARM) architecture. After performing an analysis of the interaction between this infrastructure and the attackers, the main contributions that can be extracted from this work are the following:

- We present a statistical analysis of the connections that took place in our honeypot, showing the geographical information about the origin of the attacks as well as the most commonly usernames and passwords used in brute force attacks, and the different remote

hosts to which the attackers tried to connect once they had obtained valid credentials.

- We perform an evaluation of the interaction of the attackers with our system, classifying the sessions established according to the sequence of commands introduced by the attackers.
- We present an analysis of the files downloaded by the attackers in our infrastructure, mainly consisting of binary files, but also bash-scripts and compressed files.
- We evaluate the similarity between each pair of collected files and classify them using N-grams together with the Jaccard index.

The remainder of this paper is organized as follows. Section 2 studies the proposals from the scientific community. Section 3 describes the methodology followed, and Section 4 presents the data analysis for the experiment that was carried out. Finally, Section 5 presents the main conclusions.

## 2 Related work

The concept of a honeypot was first introduced more than two decades ago when the first worms started to spread through Windows and Linux systems [2]. Honeypots are monitored systems which are exposed to the Internet with the aim of obtaining information about attacks that are occurring in real time. These systems can be classified as low, medium and high interaction depending on the functionality that is available for attackers [3]. Low and medium interaction systems do not present a complete system to the attacker (i.e. lack of commands, static file system or fixed command outputs [4]), while high interaction systems do provide a complete system to attackers and, therefore, it makes it difficult to fingerprint the honeypot based on its interaction or the tools that are available [5].

Nowadays, one of the main attack trends is targeting IoT devices, for the most part because a substantially number of these devices are more focused on providing new features to users rather than providing security or privacy measures [6] for protecting themselves and the data that they handle. In addition, since they are limited devices in terms of resources, it is highly unusual to find them using additional security measures such as AntiVirus (AV) or Intrusion Detection Systems (IDS) [7]. Due to this insecure nature, the research community, as well as the industry itself, use honeypots for detecting new threats and learning about the tactics, techniques and procedures used by attackers against these devices.

Pa et al. [8] designed a honeypot focused on Teletype Network (Telnet) attacks by combining a low-interaction honeypot with a sandboxing system. This way, when a command is unknown, it is sent to the sandbox in order to give a reliable response to the entered command, and it is stored for future requests. Another solution proposed by Šemić et al. [9] is a low-interaction honeypot for the Telnet protocol. The honeypot has two

frontends, one dedicated to manual attacks, simulating some of the commands and components of a real system, and one specifically designed to respond to Mirai malware.

Other protocols used by IoT devices have been studied by Wang et al. [10], which proposed a high-interaction honeypot for Message Queuing Telemetry Transport (MQTT) and Extensible Messaging and Presence Protocol (XMPP) modules, while device emulation was performed via a Representational State Transfer (REST) Application Programming Interfaces (API). Luo et al. [11] designed a honeypot with intelligent interaction based on the responses received from other real IoT devices which it actively scans and sends requests logged from previous attacks, and combines it with the use of a reinforcement learning algorithm to give the best possible response to attackers.

Vetterl et al. [12] proposed a high-interaction honeypot for capturing attacks on Customer Premise Equipment (CPE) and IoT devices through emulation of their firmware. The honeypot is designed to obtain information on how the system is compromised and once the attack vector is known it is blocked.

Tambe et al. [13] introduced the idea of making use of a high-interaction honeypot using Virtual Private Network (VPN) tunnels so that a physical IoT device can be listening in different geographic locations, simulating multiple devices.

Cowrie [14] is an open-source honeypot that was created as a continuation and extension of Kippo [15]. It supports the Telnet and Secure SHell (SSH) protocols and, although it was initially designed as a medium-interaction honeypot, it can be used as a high-interaction honeypot allowing the Cowrie logging system to be used transparently with real or virtual devices. Fraunholz et al. [16] uses Cowrie as a medium-interaction honeypot to perform statistical and behavioral analysis on incoming attacks.

In summary, there are different studies made by the research community that use honeypots to analyze the different attacks on protocols or services used in the IoT. Unlike most related work, we used a high-interaction honeypot to collect attacks on Telnet and SSH services, where we classified the attackers' command sessions and downloaded malware samples to spread through these devices.

## 3 Methodology

In order to learn how SSH and Telnet services are exploited for gaining access to IoT devices and perform attacks through them, we deployed a high-interaction honeypot and monitored the actions and accesses carried out by exploiters. The methodology followed in this experiment, which is shown in Figure 1.1, is explained in this section. Firstly, we describe the architecture of the deployed system and then we detail the configuration of the machines which simulated vulnerable devices. Finally, we present the analysis carried out in this study.

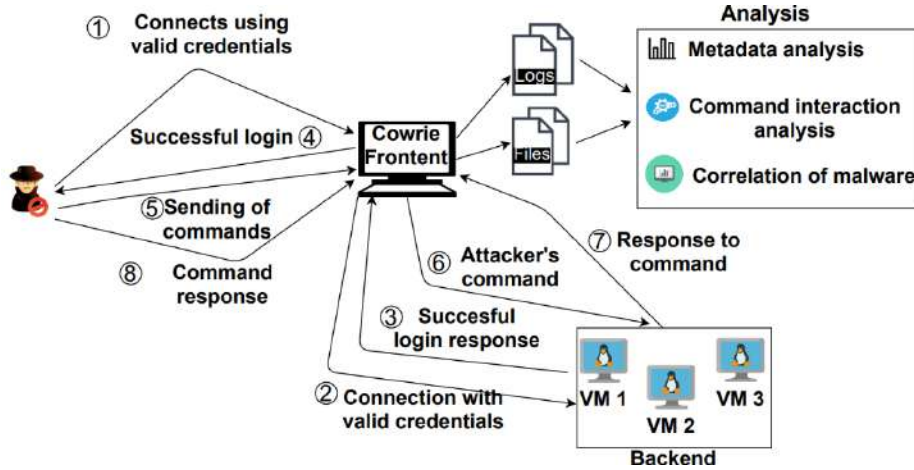


Figure 1: Outline of the methodology followed in this work.

### 3.1 System Architecture

The architecture consisted of a private server that was deployed in Germany. We hardened the server in order not to allow unauthorized access to the services in this machine, and, after that, We deployed the honeypot on this machine and redirected the connections directed to the SSH and Telnet services to the internal ports used by Cowrie. We configured Cowrie to work in proxy mode, redirecting the traffic that reached these services to virtual machines compatible with QEMU [17]. This allows Cowrie to become a high-interaction honeypot because attackers are dealing with a complete system rather than an emulation of the file system and certain commands, thus making it more difficult for them to notice that they are not compromising a real system.

For user authentication, we only considered the root user and the top 10 most common passwords used by malware targeting IoT devices [18] as valid ones. This way, when an attacker entered valid credentials, Cowrie performed a successful login in the virtual machine, returned the response from the server of the virtual machine and started acting as a proxy, redirecting the input and output of commands between the attacker and the virtual machine. The proxy mode can be configured to redirect traffic to virtual machines or devices outside Cowrie, or Libvirt compatible virtual machines can be included and Cowrie can take care of deploying and restarting the machine when needed through this toolkit.

### 3.2 Virtual machine configuration

For the creation of the virtual machine used to emulate a real system, we used Buildroot, which automates the process of creating a Linux environment for embedded systems. Using this tool, we were able to build a Linux system for the ARM architecture by cross-compiling, so that, once that an attacker has gained access to the system and performs a reconnaissance of the environment, they

find that the architecture being emulated is the most used one for IoT devices.

For the Linux system compiled, we included compatibility for the old application interface (OABI), allowing the execution of binaries created for older ARM architectures. Finally, we included different tools such as BusyBox, Perl, and Python, as well as SSH and Telnet servers to allow Cowrie to connect in its proxy mode.

Once the machine was built with Buildroot, we obtained the kernel that would be emulated and a file system and utilities that resemble those that an attacker would find on a real device. The system generated was emulated using QEMU, and, in order for Cowrie to manage the virtual machine, it was necessary to generate an Extensible Markup Language (XML) file containing the configuration options that QEMU needed to emulate the machine.

### 3.3 Data analysis

In order to evaluate the actions carried out by the attackers, the metadata that could be extracted from the stored logs was thoroughly studied. This analysis was divided into three different tasks: the inspection of the data associated with the connections, the examination of the interaction that the attackers made with the system, and the study the files that were downloaded in our honeypot.

**Metadata analysis.** In this phase, we analyzed the data associated with the connections made to our honeypot system. Firstly, we focused on the connection attempts made throughout the experiment, and then broke the data down into days of the week. Secondly, we evaluated all the login attempts that were made, also studying the username and password combinations most frequently used by the attackers. Thirdly, we studied the attacks that were attempted through the SSH feature known as port forwarding. This SSH feature allows the redirection of any Transmission Control Protocol (TCP) port and the sending of data via SSH, allowing, for example, accessing

geolocation-restricted content, bypassing firewalls, etc. Therefore, the server would act similarly to a proxy and the connection data recorded on the target host would be from the SSH server. Finally, we analyzed the origin of the attacks by consulting the geolocation of the Internet Protocol (IP) addresses that interacted with our honeypot using public IP location services [19], [20].

**Interaction analysis.** To analyze and classify the interaction, we extracted all the command sessions for each of the IP addresses. Then, we cleaned duplicated sessions from the same IP, i.e. sessions that are exactly the same and therefore have been created by bots that made another connection to the system and performed exactly the same tasks as in other connection. After this, we standardized the commands by eliminating specific semantics that can be variable and still be the same command. For this purpose, we replaced the following variables by constant values using pattern search and regular expressions:

- IP addresses and Uniform Resource Locators (URLs)
- Names of downloaded files or scripts
- Payload hardcoded in commands
- SSH keys
- Inserting users
- Replacing non-existing BusyBox commands

Once the commands entered had been standardised, we separated each one of them. In addition, those symbols that allow different commands to be chained together in the same order, such as the semicolon, were removed. After this cleaning and standardisation stage, each session was composed of an ordered vector where each element represented a command.

For measuring the similarity between sequences and performing the classification, we use the cosine similarity between two vectors [21]. Given to sequences A and B, the first step is to transform each one of them into a vector, obtaining  $\vec{A} = \{a_1, a_2, \dots, a_n\}$  and  $\vec{B} = \{b_1, b_2, \dots, b_n\}$ , where  $a_i$  and  $b_i$  represent the number of times in which the command in position  $i$  appears in the sequence, and  $n$  is the total number of different commands in the whole set of sequences that are being compared.

Finally, given two vectors of command sessions, we compute the cosine similarity as follows:

$$\text{similarity}(A, B) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \times \|\vec{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

The obtained result is in the interval  $[0, 1]$ , with 1 meaning a perfect similarity and 0 a nonexistent one. In other contexts, this interval can be  $[-1, 1]$ , but in this experiment it is not possible to obtain negative values for the vectors.

**Analysis of downloaded samples.** For the analysis and clustering of the downloaded files in the honeypot, we used the tool proposed in [22], which automatizes the analysis and feature extraction process of pieces of malware from various architectures. For the clustering and classification tasks, we followed a dynamic approach, due to the fact these might be samples from different architectures, so a static study based on opcodes would not be able to perform an interarchitecture comparison. Using

this tool, we extracted the sequence of system calls (syscalls) of size  $N$  for each sample, also known as N-grams. For example, for the following vector of syscalls: [execve, time, getpid, getppid], the set of n-grams of size 2 that will be obtained is: (execve, time), (time, getpid) and (getpid, getppid). Once that the N-grams from the different samples were extracted, we calculated the similarity using the Jaccard index [23], which allowed us to determine the similarity between two sets in the following way:

$$\text{Jaccard\_index}(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|},$$

where the numerator represents the subsets (N-grams) found in both samples and the denominator indicates all unique subsets between samples. The result is a value between 0 and 1 representing the similarity between two sets.

## 4 Data Analysis

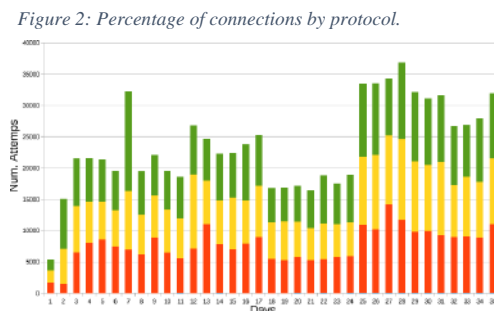
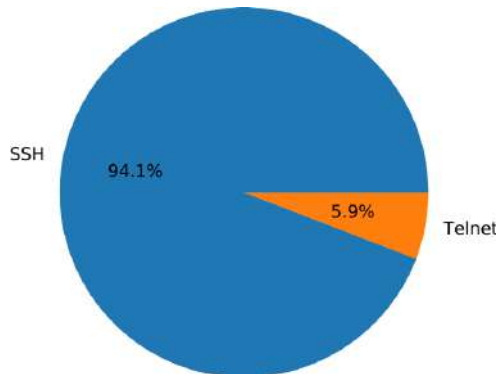
Our honeypot system ran for a period of 35 days. During this time, the actions carried out on the system were monitored and the corresponding data was collected. This section presents the results that can be extracted from this experiment.

### 4.1 Metadata analysis

In this section we analyze the information associated with the connections or connection attempts that were logged at the honeypot. Firstly, we perform a study based on the number of attempts and the time stamp at which they occurred. Secondly, we analyze the login attempts. Thirdly, we analyze the connection attempts via SSH tunnels to other hosts, and finally we analyze the origin of the IP addresses that interacted with the honeypot.

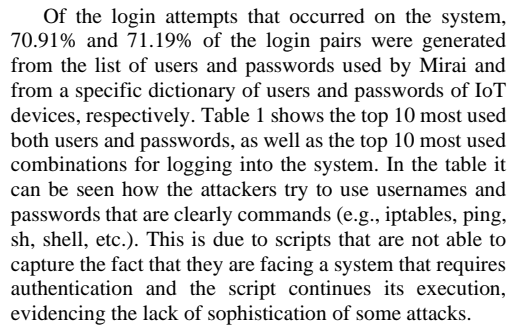
**Analysis of connections.** We analyze the timestamp of the connection attempts recorded by the honeypot using Spanish local time (Greenwich Mean Time +1). In the thirty-five days of the experiment, the system captured a total of 830,053 connection attempts. Figure 2 represents the number of attempts per protocol. It can be seen that most of the connections were through the SSH protocol, reaching a total of 781,339 connection attempts. Which is more noticeable from this data is that the number of connections captured via the Telnet protocol (48,714) is so low, as for years it was the preferred protocol for brute-force attacks on IoT devices.

As for the timeline of the connection attempts captured, Figure 3 shows the attempts that occurred on each of the days of the experiment. Red represents those attempts that occurred between 00:00 and 08:00, yellow represents those that occurred between 08:00 and 16:00, and green represents those that occurred between 16:00 and 00:00. Looking at the results, it can be seen that, on average, there were 20,000 attempts per day, with this figure even exceeding 30,000 on some days. In general, in terms of time periods, the number of attacks remains uniform and does not show any tendency towards a specific one. This indicates that most of the attacks were



automated and not focused on out-of-office hours such as the 00:00 to 08:00 period.

Figure 4 shows the connection attempts that were made on each day of the week. It can be seen that the number of attempts is very similar for each day and, although the peak is reached on Sunday, it does not seem to follow any trend.



It can be seen that brute-force attacks on SSH and Telnet credentials are still one of the main methods of finding vulnerable systems and that the trend has continued since the release of Mirai, such methods allowing attackers to compromise systems quickly without investing too much effort or money in searching for vulnerabilities or zero-days.

**Direct TCP-IP connection analysis.** We analyzed the connection attempts that allowed attackers to create TCP sessions in an SSH tunnel. In total, there were 1,278,006 connection attempts using this SSH feature, of which 33,804 were unique targeted hosts. It is important to note that all of these connection attempts were made by only 96 attackers or bots. Figure 5 graphically shows the top 30 hosts to which the most connection attempts were made. The red nodes represent the IPs of the attackers, while the blue nodes represent the hosts to which they tried to connect to. The edges represent whether there were any connection attempts between the attacker IP and the

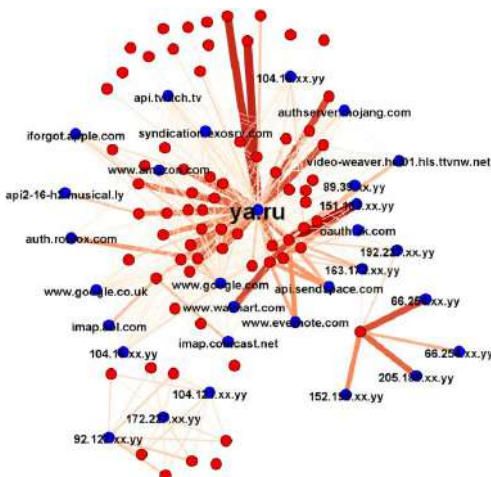


Figure 5: Connection attempts via ssh direct TCP-IP. The blue nodes represent the hosts or addresses they attempted to connect to and the red nodes represent the attacker's ip address. The edge weight indicates the number of connections from node A to node B.



Table 1: Top 10 users, passwords and combinations of both most used by attackers.

Users	Total	Passwords	Total	Pairs	Total
root	640,170	admin	411,542	root/admin	410,365
admin	16,006	root	126,693	root/root	124,801
enable	4,295	123456	9,781	enable/system	4,293
sh	4,200	123	4,401	sh/shell	4,185
test	2,927	system	4,394	ping ; sh//bin/bu..	2,002
user	2,596	shell	4,198	root/54321	1,888
ping ; sh	2,002	/bin/busy...	3,549	admin/888888	1,882
ubuntu	1,921	888888	2,771	root/8888	1,825
postgres	1,836	5555	2,590	root/5555	1,824
Iptables -F	1,547	password	2,095	root/11111111	1,821

targeted node, and the thickness of the edges varies according to the number of times the attacker tried to connect to the targeted host through our SSH server. It should be noted that no connection was made from our honeypot to the outside. Also, those targeted hosts to which they tried to connect directly through their IP address were anonymised. As can be seen in the image, some well-known domains appear, such as Google, Amazon, Evernote or the Russian search engine 'ya.ru', to which 30.5% of the total connection attempts belong to. Making requests to known domains could be a way to check whether the redirection is working correctly and whether they are in a honeypot.

In addition, we analysed the destination ports to which the requests were directed. Table 2 shows the top 10 ports and indicates that most of the requests were directed to connections via the Hypertext Transfer Protocol (HTTP) and the Hypertext Transfer Protocol Secure (HTTPS). The rest of the connections in the top 10 are mainly related to sending and receiving email and the different protocols used for this. Therefore, they were trying to connect to Simple Mail Transfer Protocol (SMTP) servers to send emails anonymously, something commonly used in spam and phishing campaigns.

Finally, an unusual port can be observed. After applying Open Source INTelligence (OSINT) [24] techniques, we observed that this port is used by the servers of RuneScape [25], an online role-playing game that is developed by Jagex. This could be used to perform a Distributed Denial of Service (DDoS) attack on the game server or to evade bans based on IP addresses in the game.

**Origin of the attacks.** In total, 9,926 unique IP addresses interacted with the system. We obtained the origin of the addresses by using location services and plotted them on a map. Figure 6 shows a heat map and the representation of the existing IP addresses in that geographical area. It can be seen that the origin of most attacks is in Asia, Europe and the United States, with China being the most common location with 28.03% of the IP addresses collected, followed by the United States (10.89%), France (5.67%), Brazil (4.49%) and South Korea (4.07%). It is worth noting that 51.42% of the IP addresses collected come from Asia, which is twice the number of IP

Table 2: Top 10 ports to which petitions were addressed.

Port	Total	Protocol information
443	526,670	https
80	508,109	http
25	176,972	smtp
993	24,855	imaps
587	22,627	smtp over tls
465	5,183	smtp over tls
43594	4,802	runescape servers
143	3,409	imap
26	1,988	smtp
2525	1,952	smtp

addresses from Europe (24.20%). It is also relevant that the origin of the actors behind the attacks is not necessarily that location specifically, as they could be using some kind of proxy, virtual private networks (VPNs), Tor or systems that have been previously compromised through malware and are acting as bots looking for other vulnerable systems to spread malware.

### 4.2 Interaction classification

This section presents an analysis of the interaction of the different attacker sessions via the SSH and Telnet protocols. As discussed in Section 4.1, only 1,534 unique IP addresses successfully logged in. Out of that entire pool of IP addresses, at least 1,402, or 91.4%, executed at least one command. In total, 4,217 sessions were established, i.e. some of these IP addresses connected to the honeypot several times.

To analyse the interaction, automated techniques were applied to classify the sessions and extract knowledge. Firstly, as described in Section 3.3, the commands entered were standardised and the most commonly used commands were extracted. Table 3 shows the top 10 commands most used by attackers. In the table, we can see some that are quite common, such as system, shell, and enable, and which are normally used on some devices such as routers to obtain shell or more privileged commands. Also, the command "echo CODE >> .file", which dumps binary code to a file, can be observed.

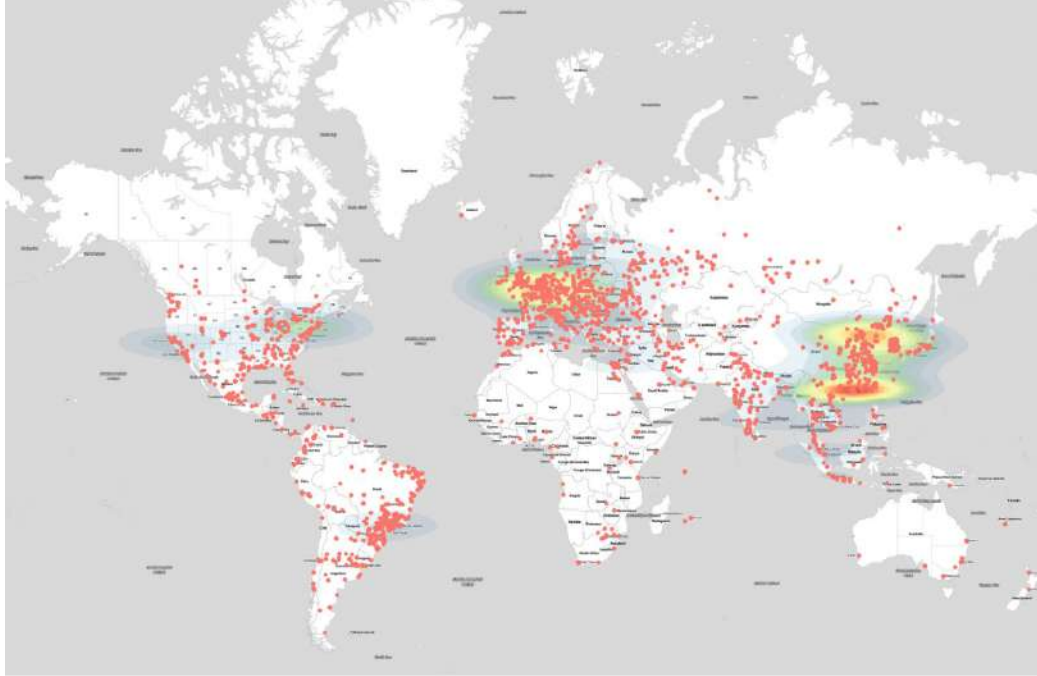


Figure 6: Origin of the attacks received in the honeypot

Table 3: Top 10 most used commands.

Command	Total
/bin/busybox BUSYBOX	6,065
sh	4,483
shell	4,197
system	4,171
enable	4,171
linuxshell	3,397
/bin/busybox cat /bin/busybox	2,629
>.file	2,099
>file	1,930
echo CODE >>.file	1,272

The most striking command is the one that was repeated most often as it is an invalid command. Attackers use the command `/bin/busybox` followed by non-existent “command” names such as CORONA, TSUNAMI, FBOT, etc., in order to find out if the previous commands have been executed correctly [26].

Secondly, the similarity between each pair of sessions was calculated using the cosine similarity. In order to consider two similar sessions, different thresholds were tested, and finally we selected one with the value of 0.9, i.e. two sessions are similar if the cosine similarity is greater than 0.9. Figure 7 shows the results of clustering sessions based on the commands entered during the session. The nodes represent the sessions and the edges connect two sessions if their cosine similarity exceeds the set threshold. It can be seen that most of the sessions are

similar to each other and that they are mostly grouped in 7-8 clusters, indicating that most of the connections were made by bots searching for vulnerable systems to download and install malware using similar tactics, techniques and procedures (TTP).

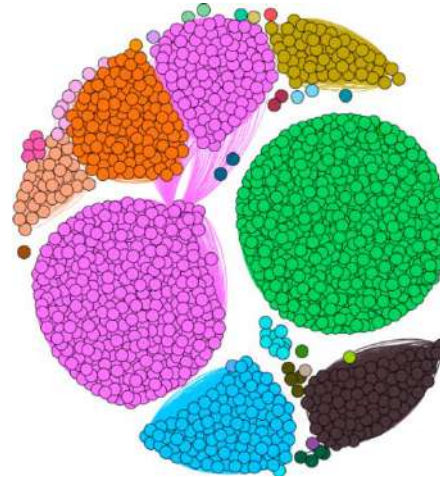


Figure 7: Clustering of sessions according to the commands entered. Each node represents a session and edges between two nodes indicate that the similarity is above the set threshold of 0.9.

### 4.3 Downloaded malware analysis

This section presents the results of the analysis of the samples collected by the honey\_pot. In total, 1,578 samples were collected, of which 710 were unique. The unique samples included 590 Linux executable binary files, 35 gzipped files, 82 bash script files and 3 perl scripts.

**Binary files.** These were executable and Linkable Format (ELF) binary files, mainly from the 32-bit ARM architecture (87.46%). The rest of the samples corresponded to other architectures such as Intel 80386, Microprocessor without Interlocked Pipeline Stages (MIPS), etc. For the classification of the samples, the syscall sequences were extracted as discussed in Section 3.3, and N-grams were extracted for each syscall sequence using four as the N-gram size. Figure 8 shows the results of the clustering of the collected samples. The nodes represent each of the samples and the edges join nodes that have a similarity greater than 80%. It can be seen that there is one cluster that stands out from the rest, and then there are small clusters or sets of connected samples.

We applied reverse engineering techniques in different samples from each cluster to confirm that the samples were clustered correctly. The samples that are clustered with any other sample based on the established similarity threshold are the following:

- **Hajime:** To this category belong 70.5% of the samples found, and it is this grouping that stands out from the rest in Figure 8. This sample is Hajime's downloader, a malware that first appeared in 2016 and spreads mainly via Telnet and vulnerabilities whose exploits it has been incorporating into its scanning and propagation module [27]. It is a botnet that communicates with its C&C through a decentralised network and its real purpose is so far unknown as it has not launched any denial-of-service (DoS) attacks. The binaries found all have the same size and are responsible for downloading the sample from the next phase [26]. Mainly, it is the same binary with only changes in the address and download port of the next stage of the malware.
- **Mirai:** This is the most popular family of malware that attacks IoT devices. It became famous because it generated the largest DDoS attack using IoT devices [28]. Its source code was leaked in 2016 and since then multiple Mirai variants have appeared. It accounts for 9.83% of the total binary files.
- **Gafgyt:** This accounts for 6.44% of the executable files collected. It is malware whose source code was leaked in 2015 and, like with Mirai, there are many variants of this malware family. Among its main features is the ability to perform various DDoS attacks [29].

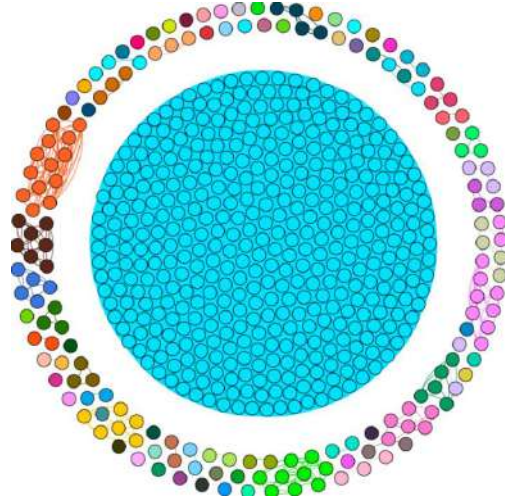


Figure 8: Clustering of the samples captured in the honey\_pot. Each node is a sample and an edge connects two nodes if they have a similarity greater than 80%.

- **Dofloo:** This is malware that allows DDoS attacks and the loading of cryptocurrency miners [30], [31], and accounts for 1.52% of the samples.
- **Xorddos:** This is malware that affects Linux-type devices and allows different types of DDoS attacks [32]. This family accounts for 2.54% of the samples.
- **Others:** The remaining samples, which were not related to any other sample on the basis of the similarity index, belong to this category.

**Bash-script.** These were script files downloaded in the sessions and designed to download malware for different architectures and execute it. All scripts work in a similar way: 1) they move to a directory where the user has permissions; 2) they download the malware for different architectures via `wget`, `ftp` or `curl`; 3) they give the downloaded file execution permissions; and 4) they execute the file with or without arguments. All files perform the same task and differ only in the IP addresses they try to connect to, the filename and the architectures supported by the malware.

**Perl-script.** The downloaded scripts were Internet Relay Chat (IRC) bots written in Perl with command execution capabilities. All three scripts were based on source code that is publicly available on the Internet, with two of them even having the same comments as the versions they were based on [33], [34].

**Compressed files.** These were files in “tar.gz” format that had a hidden folder with the name `rsync`. The sum of all the unzipped files was 979, 106 of which were unique. There were 34 shared libraries, 31 executable files and 40 script files. In general, they were different versions of malware designed to mine cryptocurrencies. Such botnets provide cybercriminals with a network for distributed

mining and financial gain, either by saving electricity bills or by obtaining cryptocurrencies [35]. Each compressed file contains several scripts that are responsible for finding and stopping the execution of any other mining malware and initiating the execution of its own malware. They also carry different versions of the executable used to perform the mining as well as the necessary libraries for ARM, x86 and x86\_64 architectures.

## 5 Conclusions

In this study, a high-interaction honeypot has been deployed for a period of one month, mimicking the behaviour of an IoT device. The data captured have been analysed yielding valuable insight of the actions carried out by the attackers once the Telnet and SSH services had been compromised. Firstly, a statistical analysis was performed based on connection attempts, authentication attempts, IP addresses and attack attempts using SSH port forwarding. Then, we analysed the interaction of the attackers with the honeypot and classified the sessions established. Finally, we analysed the different files that were downloaded on our honeypot system by classifying the ELF format binary files.

The results show that most of the sessions established were conducted in an automated manner by bots searching for brute-force vulnerable servers on which to install and execute their malware. Also, different variants of known malware families were found that are still operating and actively look for new systems to add to their botnet. In addition, different variants of malware were found that aim to use these devices for cryptocurrency mining.

With this work we have been able to gain a better understanding of the actions that attackers take when targeting vulnerable services, thus providing the research community with valuable knowledge on the behaviour of cybercriminals in the IoT, which is one of the main issues currently under study due to their importance for users and their data.

## ACKNOWLEDGMENTS

This work has been supported by the MINECO and European Commission (FEDER funds) under project RTI2018-098156-B-C52, by the JCCM under the project SB-PLY/17/180501/000353, by the Spanish Education, Culture and Sports Ministry under grant FPU 17/03105, and by the University of Castilla-La Mancha under the contract 2021-POST-20518.

## References

- [1] Dan Demeter and Marco Preuss and Yaroslav Shmelev, 'IoT: a malware story - Securelist'. 2019. [Online]. Available: <https://securelist.com/iot-a-malware-story/94451/>
- [2] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, 'A survey on honeypot software and data analysis', *ArXiv Prepr. ArXiv160806249*, 2016.
- [3] W. Fan, Z. Du, D. Fernández, and V. A. Villagrà, 'Enabling an Anatomic View to Investigate Honeypot Systems: A Survey', *IEEE Syst. J.*, vol. 12, no. 4, pp. 3906–3919, 2018, doi: 10.1109/JSYST.2017.2762161.
- [4] L. Alex Turing and Y. Xu, 'New Threat: ZHtrap botnet implements honeypot to facilitate finding more victims'. 2021. [Online]. Available: <https://securelist.com/iot-a-malware-story/94451/>
- [5] A. Vetterl and R. Clayton, 'Bitter Harvest: Systematically Fingerprinting Low- and Medium-interaction Honeypots at Internet Scale', Baltimore, MD, Aug. 2018. [Online]. Available: <https://www.usenix.org/conference/woot18/presentation/vetterl>
- [6] M. Frustaci, P. Pace, G. Aloï, and G. Fortino, 'Evaluating Critical Security Issues of the IoT World: Present and Future Challenges', *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2483–2495, 2018, doi: 10.1109/JIOT.2017.2767291.
- [7] kaspersky, 'Pushing the limits: How to address specific cybersecurity demands and protect IoT'. 2022. [Online]. Available: <https://www.kaspersky.com/blog/iot-report-2022/>
- [8] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, 'IoT POT: Analysing the rise of IoT compromises', 2015.
- [9] H. Šemić and S. Mrdović, 'IoT honeypot: A multi-component solution for handling manual and Mirai-based attacks', in *2017 25th Telecommunication Forum (TELFOR)*, 2017, pp. 1–4. doi: 10.1109/TELFOR.2017.8249458.
- [10] M. Wang, J. Santillan, and F. Kuipers, 'ThingPot: an interactive Internet-of-Things honeypot', *ArXiv Prepr. ArXiv180704114*, 2018.
- [11] T. Luo, Z. Xu, X. Jin, Y. Jia, and X. Ouyang, 'Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices', *Black Hat*, vol. 2017, 2017.
- [12] A. Vetterl and R. Clayton, 'Honware: A virtual honeypot framework for capturing CPE and IoT zero days', in *2019 APWG Symposium on Electronic Crime Research (eCrime)*, 2019, pp. 1–13.
- [13] A. Tambe *et al.*, 'Detection of threats to IoT devices using scalable VPN-forwarded honeypots', in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, 2019, pp. 85–96.
- [14] M. Oosterhof, 'Cowrie honeypot'. Cowrie, Mar. 2021. Accessed: Mar. 08, 2021. [Online]. Available: <https://github.com/cowrie/cowrie>
- [15] U. Tamminen, 'Kippo honeypot'. 2014. Accessed: Mar. 09, 2021. [Online]. Available: <https://github.com/desaster/kippo>
- [16] D. Fraunholz, D. Krohmer, S. D. Anton, and H. D. Schotten, 'Investigation of cyber crime conducted by abusing weak or default passwords with a medium interaction honeypot', in *2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security)*, 2017, pp. 1–7.
- [17] F. Bellard, 'QEMU: A generic and open source machine emulator and virtualizer'. 2020. Accessed: Feb. 14, 2020. [Online]. Available: <https://www.qemu.org/>

- [18] A. L. Johnson, 'Endpoint Protection - Symantec Enterprise'. 2016. Accessed: Mar. 09, 2021. [Online]. Available: <https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=ed1463a7-41ed-4b2f-adfd-97ccacb9d593&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68>
- [19] IP-API.com, 'Geolocation API'. 2022. Accessed: Mar. 05, 2021. [Online]. Available: <https://ip-api.com/>
- [20] shodan, 'Shodan is the world's first search engine for Internet-connected devices.' 2022. Accessed: Jul. 25, 2020. [Online]. Available: <https://www.shodan.io/>
- [21] B. Li and L. Han, 'Distance weighted cosine similarity measure for text classification', in *International conference on intelligent data engineering and automated learning*, 2013, pp. 611–618.
- [22] J. Carrillo-Mondejar, J. M. Castelo Gomez, C. Núñez-Gómez, J. Roldán Gómez, and J. L. Martínez, 'Automatic Analysis Architecture of IoT Malware Samples', *Secur. Commun. Netw.*, vol. 2020, p. e8810708, Oct. 2020, doi: 10.1155/2020/8810708.
- [23] S. Fletcher and M. Z. Islam, 'Comparing sets of patterns with the Jaccard index', *Australas. J. Inf. Syst.*, vol. 22, Mar. 2018, doi: 10.3127/ajis.v22i0.1538.
- [24] J. Pastor-Galindo, P. Nespoli, F. G. Mármol, and G. M. Pérez, 'The Not Yet Exploited Goldmine of OSINT: Opportunities, Open Challenges and Future Trends', *IEEE Access*, vol. 8, pp. 10282–10304, 2020, doi: 10.1109/ACCESS.2020.2965257.
- [25] Jagex, 'RuneScape Online Community - Forums, News, Events and more', *RuneScape*. 2022. Accessed: Mar. 05, 2021. [Online]. Available: <https://www.runescape.com/community>
- [26] S. Edwards and I. Profetis, 'Hajime: Analysis of a decentralized internet worm for IoT devices', Rapidity Networks, 2016.
- [27] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, 'Measurement and analysis of Hajime, a peer-to-peer IoT botnet', 2019.
- [28] M. Antonakakis *et al.*, 'Understanding the mirai botnet', in *26th USENIX security symposium (USENIX Security 17)*, 2017, pp. 1093–1110.
- [29] A. Marzano *et al.*, 'The Evolution of Bashlite and Mirai IoT Botnets', in *2018 IEEE Symposium on Computers and Communications (ISCC)*, 2018, pp. 00813–00818. doi: 10.1109/ISCC.2018.8538636.
- [30] K. Angrishi, 'Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV): IoT Botnets', *ArXiv170203681 Cs*, Feb. 2017, Accessed: Mar. 08, 2021. [Online]. Available: <http://arxiv.org/abs/1702.03681>
- [31] S. Gatlan, 'Exposed Docker APIs Abused by DDoS, Cryptojacking Botnet Malware', *BleepingComputer*. 2019. Accessed: Mar. 08, 2021. [Online]. Available: <https://www.bleepingcomputer.com/news/security/exposed-docker-apis-abused-by-ddos-cryptojacking-botnet-malware/>
- [32] M. D. Donno, N. Dragoni, A. Giaretta, and A. Spognardi, 'Analysis of DDoS-capable IoT malwares', in *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2017, pp. 807–816. doi: 10.15439/2017F288.
- [33] H3LLS1NG, 'Shadow-Network/perl-scripts'. Shadow-Network, Oct. 2020. Accessed: Mar. 08, 2021. [Online]. Available: <https://github.com/Shadow-Network/perl-scripts>
- [34] Phl4nk, 'Ircbot'. Feb. 2018. Accessed: Mar. 08, 2021. [Online]. Available: <https://github.com/phl4nk/ircbot>
- [35] J. Carrillo-Mondéjar, J. L. Martínez, and G. Suarez-Tangil, 'Characterizing Linux-based malware: Findings and recent trends', *Future Gener. Comput. Syst.*, vol. 110, pp. 267–281, 2020, doi: <https://doi.org/10.1016/j.future.2020.04.031>.



## CHAPTER 3

# On how VoIP attacks foster the malicious call ecosystem

- **Title:** On how VoIP attacks foster the malicious call ecosystem.
- **Authors:** J. Carrillo-Mondéjar, J.L Martínez and G. Suarez-Tangil.
- **Type:** Journal paper.
- **Journal:** Computers & Security.
- **Publisher:** Elsevier.
- **ISSN:** 0167-4048.
- **Status:** Published.
- **Publication date:** August 2022.
- **Volume:** 119.
- **DOI:** 10.1016/j.cose.2022.102758
- **JCR IF/ranking:** 5.105/Q2 (JCR2021).



Contents lists available at ScienceDirect

Computers &amp; Security

journal homepage: [www.elsevier.com/locate/cose](http://www.elsevier.com/locate/cose)

# On how VoIP attacks foster the malicious call ecosystem

J. Carrillo-Mondéjar<sup>a,\*</sup>, J.L. Martínez<sup>a</sup>, G. Suarez-Tangil<sup>b</sup><sup>a</sup> Universidad de Castilla-La Mancha, Albacete, Spain<sup>b</sup> IMDEA Networks Institute, Madrid, Spain

## ARTICLE INFO

### Article history:

Received 23 September 2021

Revised 7 May 2022

Accepted 14 May 2022

Available online 16 May 2022

### Keywords:

Honeypots

VoIP

Cybercrime

Telephony fraud

Robocalls

Attacks

Underground economy

## ABSTRACT

Switched telephone networks are a key and ubiquitous infrastructure. Recent technological advances have integrated modern and inexpensive systems into these networks in order to use the Internet to place calls via Voice over IP (VoIP). The evolution of this technology has also led to an increase in the number and sophistication of the techniques used by criminals to commit fraud. Specifically, with the emergence of VoIP, attackers can now adapt tools commonly used by cybercriminals, such as botnets, to make their attacks more complex and insidious. For example, through bots they can dial multiple numbers automatically, enabling them to target a greater number of victims, and do so more quickly. While recent studies have shed light on how certain parts of this ecosystem work, it is still unclear how attacks on VoIP systems contribute to this type of fraud. This paper presents a novel VoIP honeypot that captures voice interactions, in addition to employing low-level telemetry. With the study of how attackers obtain access to our honeypot and the actions they perform, we present an overview of the most prevalent types of fraud used in this ecosystem, including unique insights into the origin of the attacks and the destination of calls made through our architecture. Finally, we analyze in depth the actions taken to study the different types of telephony fraud.

© 2022 The Author(s). Published by Elsevier Ltd.

This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>)

## 1. Introduction

The invention of the telephone is one of the most important accomplishments of humankind, eliminating the physical barriers of time and location and allowing two people from places miles apart to establish immediate communication. Although it was not designed to be used as an attack vector, unfortunately criminals found a way of using it as such in order to make unwanted or malicious calls to perpetrate extortion, scams and spam campaigns. These calls have been a long-standing and challenging problem, and are responsible for yearly losses of tens of millions of dollars worldwide (Li et al., 2018), besides being very disturbing for telephone users, who feel their privacy is violated.

The recent development of Voice over IP (VoIP) has caused telephone communications to be transferred to a more modern medium, namely the Internet, making it more convenient and cheaper for people to contact each other. These benefits have been well-received not only by ordinary users, but also by cybercriminals, who can now reduce the cost and operational complexity

of their telephony campaigns, and this technology allows them to place automated calls with little effort. These calls are typically made in the form of bot-calls, which are carried out by simple software using dialer equipment to generate vast numbers of calls to a given (or randomly chosen) list of phone numbers. Their aim is to make contact with an active recipient and trick them into performing the desired action, ultimately making them fraud victims. Generally, the instructions are transmitted via a pre-recorded message but, in some cases, they can eventually be assigned to a human agent for further interaction, although this becomes a limiting factor, since human agents may not have time to attend to all the connected calls.

In comparison with other attacks such as email spam, voice spam calls are significantly more disturbing because they require immediate attention; when the phone rings, the recipient must decide whether to accept the call and listen to it just judging by the caller ID, which is often “unknown” or even spoofed (Tu et al., 2019). Even if the user ignores or declines the call, spammers can send a prerecorded audio message straight to the user's voice-mail, thus also having the static factor of email spam. In addition, these attacks feel more personal, since, typically, people are less inclined to give their phone number than their email address, so when an attacker is able to contact them, they gain an advantage over the recipient, who normally feels confused. Furthermore, most

\* Corresponding author.

E-mail addresses: [javier.carrillo@uclm.es](mailto:javier.carrillo@uclm.es) (J. Carrillo-Mondéjar), [jose Luis.martinez@uclm.es](mailto:jose Luis.martinez@uclm.es) (J.L. Martínez), [guillermo.suarez-tangil@imdea.org](mailto:guillermo.suarez-tangil@imdea.org) (G. Suarez-Tangil).<https://doi.org/10.1016/j.cose.2022.102758>0167-4048/© 2022 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>)



email spam is systematically filtered out by anti-spam algorithms, whereas, at the time of drafting this proposal, this type of countermeasures do not exist in telephony fraud.

Related works attempt to address the problem of malicious calls in two ways: i) statically prior to the call being established; or ii) dynamically during the call. On the one hand, several approaches have performed call request header analysis (Tu et al., 2016). However, this type of measures are easily evaded (e.g., by using a spoofed caller ID (Tu et al., 2019)). On the other hand, recent works have leveraged automated approaches to interactively deal with unsolicited calls. Works such as Li et al. (2018) propose machine learning as a means to prevent malicious calls tailored to the users using features that model the normal behavior expected (e.g., weekday and time). The authors in Sahin et al. (2017) study the effectiveness of an interactive voice response system called Lenny. One of the key lessons in this work is that chatbots can be used for *scambaiting*<sup>1</sup> fraudsters. Finally, other authors have carried out systematic studies to understand technical support scams, uncovering the call centers underpinning this fraud using Web crawlers to find scammers, domains and phones on the Internet (Miramirkhani et al., 2016).

While recent studies have shed light on how certain parts of this ecosystem work, it is still unclear how attacks on VoIP systems contribute to this type of fraud. In this paper, we show how fraudsters leverage VoIP systems to build their infrastructure, and we provide an overview of the current status of VoIP systems and the risks arising when the system itself, or the credentials of its users, is compromised. Some of our findings include:

- Attackers use our system to carry out different types of telephony fraud as well as actions that allow them to obtain greater benefits. In particular, we identify both Toll Evasion and Revenue Share as being two of the most prevalent types of fraud. We also report a large number of cases in which attempts are made to call personal numbers (both landlines and mobiles), which suggests that attacks on VoIP systems are frequently used for fraudulent purposes such as scams or spam such as those described in Sahin et al. (2017). Our work is the first to provide a quantitative and qualitative study of how fraudsters use compromised VoIP systems for *fun and profit*.
- We uncover the tricks fraudsters use to evade restrictions. For instance, there are attempts to call numbers using different prefixes to avoid dial-plan restrictions. By causing an error in the parsing of these numbers, calls to restricted destinations are permitted. Calls to wrong numbers are also used to check fingerprint honeypots, e.g., when an invalid number gives them a ringtone. Learning from these tricks enables us to propose countermeasures against VoIP attacks.
- We find that while a few attacks are random, many others are aware of the context surrounding the target of the attack, namely the *callee*. In particular: (1) fraudsters are aware of the mechanisms used by reputation systems to block connections from TOR nodes or known VPNs, and consequently 50% of IP addresses come from compromised servers that belong to hosting companies; and (2) fraudsters try to maximize the success of their attacks by calling during office hours or hours when there are usually people at home.

The rest of the paper is organized as follows. Section 2 provides an overview of the system, the experiments carried out and the ethical considerations. Section 3 presents the data analysis from our experiments. Section 4 describes the limitations and key findings of our study. Section 5 discusses the proposals from the com-

munity regarding malicious calls. Finally, the conclusions are presented in Section 6.

## 2. Methodology

To understand how VoIP services are misused, we deployed a honeypot and monitored the accesses and actions that took place through it. Fig. 1 shows the methodology followed in this work. First, we describe the architecture of our system and the experimental setting. Next, we describe the vulnerable user accounts that were added to our system so that attackers could gain access through them. Finally, we describe the analysis carried out and the ethical principles followed in this work.

### 2.1. System overview

Our honeypot runs on top of a custom infrastructure with a number of VoIP accounts. The architecture consists of a Virtual Private Server (VPS) hosted in Germany, and we fortified this server to prevent unauthorized access to services that were not relevant to our study. We built an Asterisk-based Private Branch Exchange (PBX) (Sangoma Technologies, 2021), which is a popular open-source communications framework. We decided to use a real server instead of an existing honeypot because honeypots tend to be more limited in terms of interaction and can be detected through simple fingerprinting (Vetterl and Clayton, 2018). We configured our server to allow any type of call from a properly authenticated user and we redirected the calls to an enabled private extension to simulate that the call was being handled correctly. All the accounts were subscribed to a specific dialplan that limited outgoing calls to a predefined set of countries and we restricted calls to premium numbers. The reason to restrict calls to some specific places is to simulate the legitimate behavior of a configured PBX via which it is only possible to call certain destinations.

We populated our VoIP system with a number of user names and passwords, as described in Section 2.2. Each of the accounts deployed belonged to a specific context on the server and was linked to one of the dialplans that were set up. Thus, accounts linked to a dialplan in the US, for instance, could only make calls to the US. As a result, all the calls that matched the established dialplan were redirected to one of our extensions to make fraudsters believe that the calls were successful. Our system then collects logs from Asterisk in order to perform a data analysis as described in Section 2.3.

### 2.2. Accounts

We designed two experiments whose aim was to differentiate how attackers gained access to our honeypot accounts.

**Weak user credentials.** We set up a set of 30 user accounts with weak passwords. The user accounts were numerical identifiers designed to reproduce a setting similar to the one used in large companies or universities. Thus, each user (phone) was identified as if it were a phone extension, and this same extension number was used as the password. We only used numerical user IDs and weak passwords in order to promptly capture automated bots running brute force attacks against VoIP services.

**Strong user passwords.** Our second experiment sets user accounts by using: i) numerical users, similar to how they were created in the previous sections but this time using randomly generated strong passwords; and ii) common users created with the concatenation of a name and surname. For the generation of random names and surnames we used Enron's public email dataset (Klimt and Yang, 2004). This dataset was made public when the investigation into the Enron company ended. We extracted all the names and surnames, and randomly generated

<sup>1</sup> The act of wasting the time and resources of scammers.

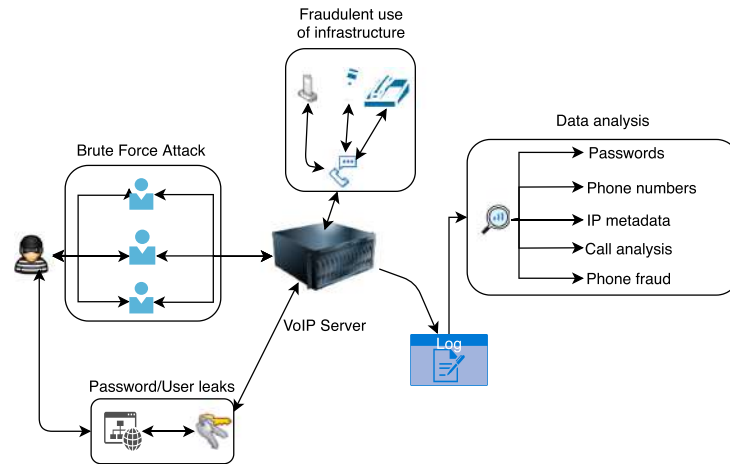


Fig. 1. Methodology.

dummy identities which we used to create users in our VoIP system. For this latter set of users, we generated the passwords by concatenating a long list of different dictionary words to simulate passwords created by the user that can be remembered.

**Account leak.** The next phase of this experiment was to leak the credentials in a controlled experiment in which we recorded the user name and the source of the leak. We chose a series of paste sites and underground forums to leak credentials since these sites are often used to publish credentials stolen from different Internet services. We leaked 70 accounts to different chosen places, with 30 of them being published on paste sites such as [pastebin.com](https://pastebin.com), [paste.ee](https://paste.ee) and [pastebin.xyz](https://pastebin.xyz). Another 20 accounts were published in underground forums such as [offensivesecurity.net](https://offensivesecurity.net) and [blackhatworld.com](https://blackhatworld.com). The remaining 20 accounts were leaked in prominent Russian underground forums. We tried to mimic the way in which information leaks from other services have occurred. As it is not a publicly known service such as gmail, facebook, etc., we also included the IP address and port of our VoIP server.

### 2.3. Data analysis

In this phase, we performed a data-driven analysis on the actions carried out in our system by the attackers and on the metadata that could be extracted from the attackers that interacted with our honeypot. In particular, our analysis first aimed to characterize the features of the IP addresses we received connections from. We then reversed the password used by the attackers when attacking the authentication mechanism of the VoIP server. Finally, we analyzed both the phone numbers and the metadata inferred from the call attempt. We describe all this in greater detail below.

**IP address analysis.** We analyzed all the metadata associated with the IP addresses that interacted with our system as well as the open ports of the systems associated with them. We analyzed the location of these addresses to discover the origin of the attack and we also checked whether these addresses appear in lists of known proxies or bots. Finally, we consulted the Autonomous System Name to which the IP addresses belonged as well as the open ports and vulnerabilities presented by the systems that were running behind them. It should be noted that no type of active scanning was carried out on these IP addresses and that

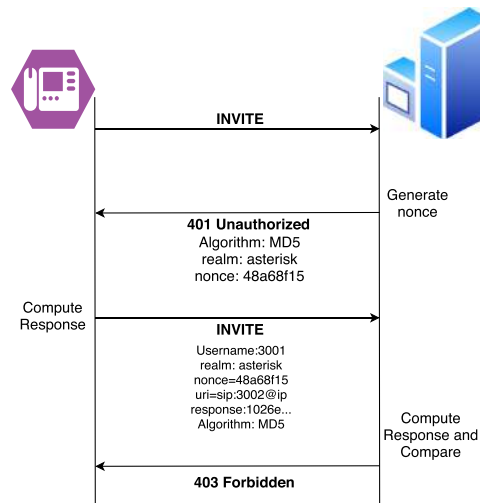


Fig. 2. Authentication message flow in the SIP protocol.

open-source tools available through the Internet were consulted (e.g., [Shodan \(2022\)](#) and IP Intelligence ([Getipintel, 2022](#))).

**Passwords.** Although in the first version of the SIP protocol the passwords travel in plain text through the network, in version 2 the SIP protocol uses a mechanism based on challenge/response to carry out user authentication ([Strand and Leister, 2011](#)). Fig. 2 shows the exchange of messages when a user fails to authenticate through the INVITE method. Unlike the REGISTER method, which registers the device and tells the server where it should direct the calls destined for that user, the INVITE method allows calls to be made without registering the device on the server.

Basically, the client sends an INVITE request to the server, which returns a message saying that it is not authorized, and in

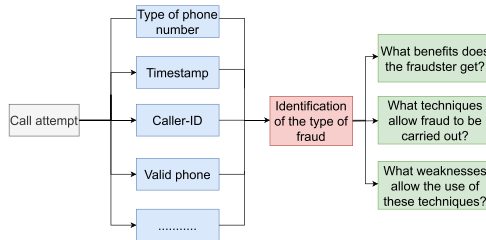


Fig. 3. Methodology followed to identify the type of fraud.

which it transmits a nonce and the realm to authenticate. The client computes the response using Algorithm 1 and sends it back to the server. Finally, the server computes the response and, if it matches the one sent by the user, it will send a correct authentication message, or in this case it will send a message that the authentication has not been carried out successfully.

$$\begin{aligned} A &= MD5(\text{user} : \text{realm} : \text{password}) \\ B &= MD5(\text{method} : \text{sip} : \text{uri}) \\ \text{hash} &= MD5(A : \text{nonce} : B) \end{aligned} \quad (1)$$

Therefore, the server never sees the plain text password used by the client, which in this case is the password used by the attackers to try to identify themselves to our server. The authentication message flow of the REGISTER method is the same, with only the method parameter changing when computing the response. By using all the data collected from the authentication messages, we obtained all the variables in Algorithm 1 except the password since it is not transmitted in plain text over the network. Therefore, we used a dictionary of the most common passwords to apply brute force and find the most frequently-used passwords on our system.

**Phone numbers.** We analyzed all the phone numbers that the attackers tried to call through our system. We parsed these numbers to extract information about whether the number was valid based on the patterns that phone numbers follow around the world. We also extracted the location of the phone number based on the country code and the area code, as well as the type of phone number (e.g., land-line or mobile).

**Call analysis and phone fraud.** We analyzed the calls that were made through our system by looking at the timestamp of the calls that were attempted in order to check whether the calls were made while taking into account the local time of the destination or whether they were automated calls that did not use this information. Therefore, on the basis of the type of call, we classified it according to the type of fraud that was being attempted through our system and the benefits that attackers could obtain through this type of fraud. For the identification of the types of fraud we rely on the taxonomy described in Sahin et al. (2017), which systematically explores the different types of fraud in telephone networks as well as the techniques and benefits obtained by fraudsters. Fig. 3 shows how we apply the taxonomy in our scenario to identify the type of fraud that is being carried out in the call attempt. Once the fraud scheme that is being attempted is discovered, we identify the possible benefits as well as the techniques and weaknesses that allow this fraud scheme to be carried out.

## 2.4. Ethics

We followed the ethical principles for Internet-mediated research in Anabo et al. (2019), which in turn stem from The Belmont Report (Miracle, 2016) and the Menlo Report (Dittrich et al., 2012). We addressed the challenge of obtaining informed consent

Table 1  
Summary of the actions performed on the VoIP server.

Type	Calls	
	Weak credentials	Strong credentials
Call attempts	306,044	713
Unique phone numbers	211,113	319
Valid phone numbers	10,731	141
Possible phone numbers	34,913	98

from unknown Internet users through the *beneficence* principle and by making a risk-benefit assessment. On the one hand, we implemented strict mechanisms to minimize risks, reducing the harm to mere *annoyance*. On the other hand, our honeypot effectively reduces the surface of the attack and prevents criminals from benefiting while depleting the resources of the attackers. We also gained an understanding of the underlying ecosystem that can allow the community to further tackle telephony scams. Finally, by granting attackers access to our VoIP we deterred them from using other compromised VoIP systems. Our risk-benefit assessment was evaluated by our Institutional Review Board (IRB), who considered our decision-making ethical and have approved this study. We next detail the measures we adopted to minimize risks to privacy.

First, we limited the usage of our VoIP system by redirecting calls to a private extension, thus preventing these calls from being forwarded to their actual recipient. Second, we restricted the scope of our experiment by limiting the number of target countries. Although no external calls were made from our system, we simulated that these were being made by calling a private extension created for this purpose, so attackers might believe that these numbers existed and try to call from other systems. Finally, for the analysis of the metadata of all the attackers of the system, no active scan was performed. Instead, third-party services (e.g., Shodan (2022)) were used to obtain information and geolocations.

## 3. Data analysis

We ran the experiment for a period of around 4 months for accounts with weak credentials. After those 4 months, we blocked access to those weak accounts and created new accounts with strong passwords, as described in Section 2.2. This experiment lasted around 2 months. During the course of both experiments, we monitored the actions performed on the VoIP server, collecting information for further analysis.

### 3.1. Overview

We deployed an Asterisk-based PBX server for our experiment. During the first period of the experiment, the server used accounts with weak credentials, while in the second part of the experiment the server used accounts with strong credentials. After parsing the information collected on our system, we detected 2209 unique IP addresses on the server. For the first period, all the accounts that were created in the VoIP system were accessed. On the other hand, out of the 70 accounts we leaked, 20 were accessed, with 12 of these having been leaked in underground forums and 8 on paste sites.

We logged the actions performed with all the accounts, which can be summarized in Table 1. We can see that the total number of call attempts is around 300 K. Unique phone numbers amount to about 200 K and represent the total number of different phones to which calls were attempted. We used the phonenumber library Drysdale (2022) to validate the phone numbers. This library maps out a taxonomy of phone number patterns of all the countries worldwide and indicates whether the phone number is valid. We considered phone numbers as possibly valid when we saw

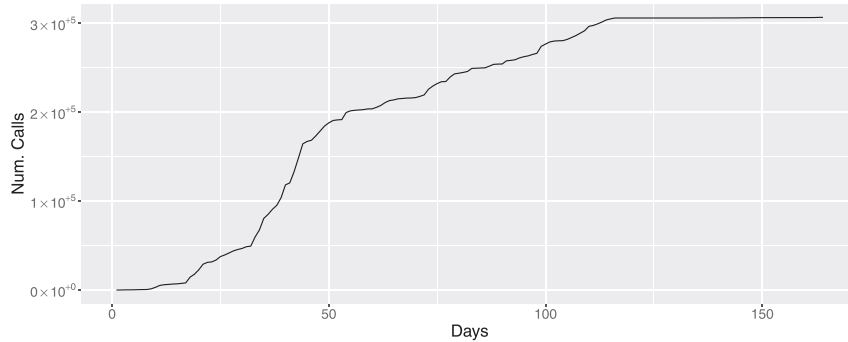


Fig. 4. Cumulative number of calls attempts (weak and strong credentials).

numbers that did not contain the country code but would be valid for a local dialplan in at least one country. For example, a number that does not contain the country code +44 (UK), but is a correct number when calling from a UK dialplan. As these numbers are often correct in different regions, we can not determine the exact destination of the call. The total number of valid and possibly valid phone numbers was around 10 K and 30 K. It is important to note that the library indicates whether the number is valid based on the dialing plan of that country, but it does not mean that this number is currently owned by any user. By observing the results, it can be seen that the number of call attempts is much higher during the time that the weak credentials experiment lasted. These calls are usually generated by bots that are scanning the Internet for active VoIP servers and are looking for valid credentials by brute force and, once they find them, they start generating calls. On the other hand, when using strong credentials, attackers could not successfully access the honeypot using brute force, and therefore we are confident that the access was through the password leaks made through forums or paste sites. As expected, the number of accesses was lower, and there may even have been curious accesses made by attackers who were interested in seeing whether they could call for free.

Regarding the call attempts, we analyzed their timestamps for each day of the experiment. Fig. 4 shows the results of this analysis.

It can easily be seen that, during the experiment with weak credentials, the number of calls increased daily while remaining constant at the end of the experiment, which is when strong passwords were used. In addition, the first calls with weak credentials occurred within a few hours of putting the server online, while with strong credentials, the first calls occurred around 20 days after leaking the credentials of the user accounts.

### 3.2. IP address analysis

In this section, we analyze the IP addresses that interacted with our system. In total, 1280 and 1188 unique IP addresses interacted with the server with weak and strong credentials, respectively. Between the two sets of IP addresses, 259 match on the two servers. Out of all those IP addresses, only 244 and 19 successfully logged in. The number of IP addresses that interacted with both servers is very similar in terms of the number of different addresses detected. However, it is important to note that the strong credentials experiment started immediately after the weak credentials one finished, and only 259 IP addresses coincide between the two experiments, which seems to indicate that, either the actors that were

Table 2

Top 20 sources and destinations of calls with weak and strong credentials. The table represents only those calls for which it was possible to extract the phone number's geographical location.

Weak credentials			Strong credentials		
Src	Dst	Calls	Src	Dst	Calls
DE	UK	822	PS	US	78
US	UK	742	PH	CU	33
NL	UK	488	PH	PH	25
NL	FR	407	PH	UK	11
NL	RU	387	PS	ES	8
NL	US	355	PS	IL	6
NL	BO	313	PH	BJ	5
NL	BQ	294	PS	UK	9
US	SG	273	US	US	3
NL	AW	273	US	UK	3
NL	LK	270	US	EG	3
NL	IQ	255	PH	US	3
NL	CW	213	PH	TN	3
NL	AM	207	PH	SI	3
NL	GT	191	PS	JE	3
DE	ZM	176	PS	CH	3
DE	RS	172	PS	FR	3
US	JP	171	PS	AT	3
NL	PE	171	DE	FR	3
NL	JP	171	CH	RU	2

actively searching for online servers changed their IP address to avoid blacklist bans, or those that might have been compromised servers detected the intrusion and closed their access.

**Location.** We depict the source of the IP address in Fig. 5. Blue points represent the origin of the IP addresses that interacted in the experiment with weak credentials, green represents the origin of the IP addresses that interacted in the experiment with strong credentials and red represents those IP addresses that matched in the two experiments.

Germany is the most common location with 21.9% of the IP addresses collected, followed by the Netherlands (20.8%), the United States (14.9%), France (12.1%) and the Palestinian Territory (10.2%). It can be seen that there is a large group of IP addresses in the same zone that tried to interact in the strong credentials experiment. This may be due to a side effect of the lack of success of the brute force attack.

We also investigated the origin and destination of the calls that were attempted through our system. We only considered those calls made to valid numbers and from which the geographical location of the telephone number could be extracted. Table 2 shows the top 20 sources and destinations of calls that were attempted

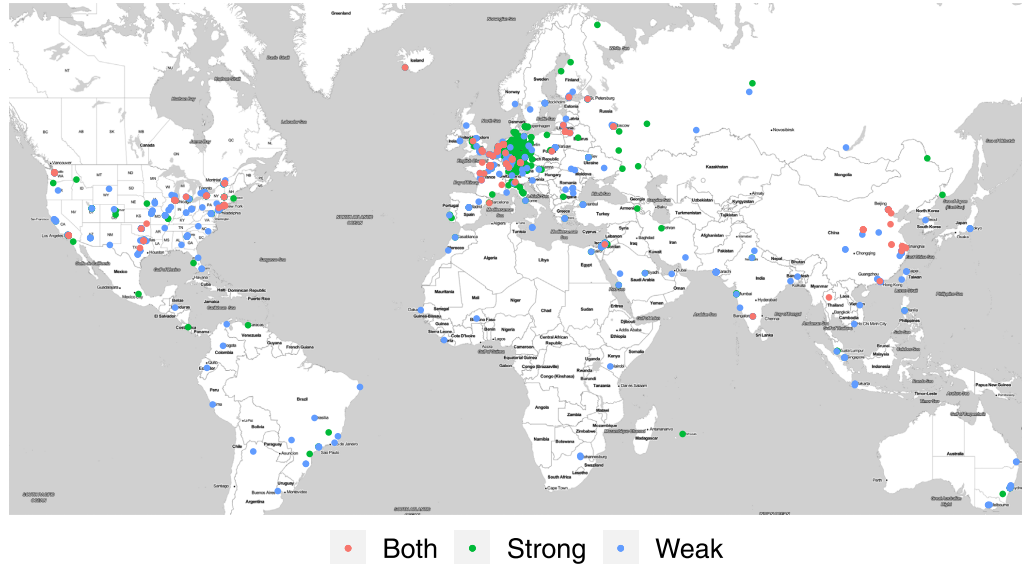


Fig. 5. Geographical map of the location of the IPs that interacted with our system.

with both weak and strong credentials. It can be seen that in the experiment with weak credentials most of the calls were attempted from the Netherlands, while with strong credentials they came from Palestine and the Philippines. It should be noted that this source may not be the actual location of the different actors, but may be hidden behind some proxy, Virtual Private Network (VPN), hostings or similar system with the IP address of those countries.

**Hostings.** We also analyzed the IP addresses through public services (e.g., VirusTotal or Shodan) to search for information and see whether there were relationships between them. Although some IP addresses were previously associated with certain malware samples and domains, there does not seem to be a clear relationship between them. However, on the basis of information about the organization or ISP to which they belong, they do indicate that certain IP addresses were related to each other. By consulting the public databases of IP address information services, we found that around 45% of the IP addresses that appeared in our system belong to hosting services. We plot a graph with the different IP addresses that belong to hosting services together with their autonomous system (AS). Fig. 6 shows the connections between the different IP addresses collected. It can be seen that large groups of IP addresses belong to the same organization (e.g., Online S.A.S or OVH), which seems to indicate that these sites may have been compromised and used by fraudsters to hide their identity or that these hostings are less restrictive when it comes to banning clients that generate fraudulent traffic. We also consulted the IP addresses via an intelligence service (Getipintel, 2022), which uses probabilistic techniques and machine learning models to determine whether an IP address is a proxy, a VPN, a hosting service, tor node, etc. Given an IP, the system returns a value of 1 if the address is explicitly banned from their lists. Otherwise it returns a value between 0 and one, indicating the probability that it is a bad IP. According to the results obtained, 477 are explicitly on their blacklists and 41 have a value greater than 0.98, indicating that these IP addresses have a high probability of being proxies, vpn's, etc.

Table 3  
Top 30 most common open ports.

Num. IP	Port	Num. IP	Port	Num. IP	Port
404	80	65	445	46	995
301	443	60	8089	45	465
289	22	58	5985	44	500
163	5060	56	143	44	137
116	3389	55	993	32	8000
92	53	49	1723	32	2000
89	21	49	110	29	5222
85	123	48	111	27	8443
75	25	47	8080	27	2087
73	3306	47	587	27	2082

**Open ports.** We analyzed the open ports of the IPs that appeared in the logs of the VoIP system. Table 3 shows the most common open ports among the IP addresses collected.

We observe that there are open ports of common services, such as HTTP, SSH, SSL, SMB, FTP or RDP. Also, one of the methods used by attackers to compromise systems is through any of these services, either because of some vulnerability or through the use of weak passwords to access through SSH. In fact, most botnets in the IoT ecosystem infect different devices through the use of weak passwords in services such as SSH or Telnet (Antonakakis et al., 2017; Kambourakis et al., 2017). Another curious circumstance that we found regarding open ports is the appearance of ports 5060 and 3389. The former is a port belonging to UDP and which is the default port used by VoIP systems. This seems to indicate that they are tracking other VoIP systems and searching for (or using) passwords from other VoIP systems to use them on their system and make calls through other users. The latter is the port used by the RDP protocol, which is used for the remote Windows desktop. The RDP protocol contains a recent vulnerability known as Blue-Keep (Microsoft, 2019), whose exploit was made public in the summer of 2019 (Proffitt and Wolf, 2019). This vulnerability allows the execution of remote code and could be used by attackers to





Fig. 6. Relations between the different IPs collected that belong to hosting services.

compromise systems and use them for their interests, such as attacking the user credentials of a VoIP system or as a proxy to carry out their attacks and not reveal their real IP address.

**Vulnerabilities.** We consulted the IP addresses through open source intelligence services (e.g. Shodan) in order to obtain the vulnerabilities detected in the services that are exposed to the Internet. Of all the information collected, we found 495 different Common Vulnerabilities and Exposures (CVEs). Fig. 7(a) shows the distribution by year of the vulnerabilities that were collected. It can be seen that there are vulnerabilities that were discovered more than 10 years ago, so it seems to indicate that not enough attention has been paid to these servers by their administrators. Also, it can be seen that they are mostly recent vulnerabilities, and that the maximum peak is reached in the vulnerabilities of 2016. It is important to note that these open source intelligence services, shows the current vulnerabilities based on the software version that is running, but do not check whether the vulnerability actually exists. We obtained information on each of the CVEs in the public database of [Cvedetails](#) (2022). Each CVE has an associated score that indicates the criticality of the vulnerability and its impact. Fig. 7(b) shows the distribution of CVEs based on the score collected for each one in cvedetails. It can be seen that most of the CVEs have a score between 4 and 8, which corresponds to

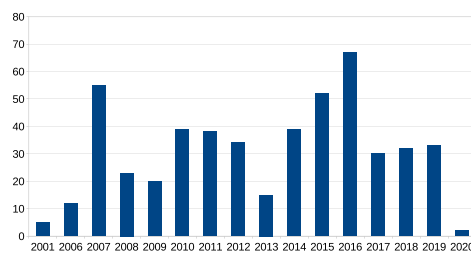
medium (4–6.9) and high (7–8.99) vulnerabilities. We can also observe that there are several CVEs with a high criticality index (9 and 10). Among the critical vulnerabilities that exist, most of them are related to the Hypertext Pre-Processor (PHP) interpreter. In addition, vulnerabilities appear in Windows systems with the Remote Desktop Protocol (RDP) and the Server Message Block (SMB), which are widely known for the impact they have had throughout the world ([s.r.o, 2017](#); [Mitre corporation, 2019](#)).

### 3.3. Passwords

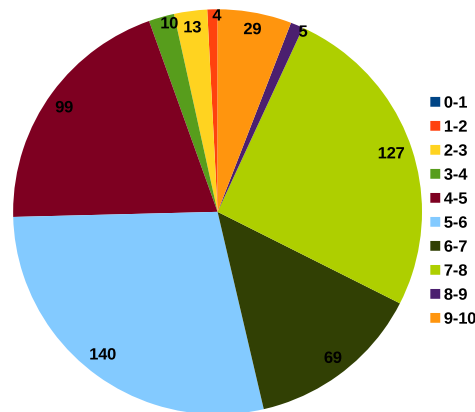
In the Asterisk security event log, you can find the hash computed to carry out the authentication, as well as the nonce sent by the server and the other parameters necessary to calculate the response, except for the password used by the user to authenticate. We parsed the security logs to find each of the fields necessary to compute the response for each of the server's failed attempts. Since we have all the data, except the password and the response hash, brute force can be applied by generating a response for a given password and checking whether the resulting hash is equal to the hash that the user sent in the authentication process. We applied brute force to each of the failed authentication attempts by using a dictionary of the 1000 most common passwords from

**Table 4**  
Top 30 most common passwords used in our system.

Rank	Passwords	Hits	Rank	Passwords	Hits	Rank	Passwords	Hits
1	1234	7944	11	abcd1234	5330	21	asdfg	2330
2	123,456	7006	12	123abc	4911	22	1111	2050
3	1000	6481	13	12,345	3501	23	8888	2023
4	password	6103	14	test123	2921	24	9999	2022
5	pass123	6032	15	qwe123	2540	25	1q2w3e	2018
6	12,345,678	5563	16	asd123	2531	26	1,234,567	2016
7	123,123	5545	17	test	2491	27	111,111	2000
8	abc123	5541	18	password123	2466	28	123,321	1995
9	0000	5442	19	admin123	2423	29	123qwe	1972
10	4321	5431	20	secret	2369	30	123,456,789	1971



(a) CVEs distributed by the year of their appearance.



(b) CVEs distributed by their criticality score.

**Fig. 7.** CVEs collected from the different IPs that interacted with our honeypot.

Miessler and Haddix (2022). In total, we managed to find 411,497 passwords for all the failed login attempts that occurred on our system (less than 1%). Table 4 shows the top 30 most commonly used passwords of all those found by brute force. The low percentage of cracked passwords could indicate that attackers are using larger dictionaries or even algorithms to generate passwords based on username. In the experiment with weak credentials we used as username and password the SIP extension and all the accounts were compromised, so it is likely that they use an algorithm to create passwords based on the extension they are trying to find.

### 3.4. Calls and fraud classification

In this section, we present the different destinations of the calls made as well as the types of fraud identified.

**Table 5**  
Types of phone numbers that attackers tried to call.

Type of phone numbers	Valid phones	Possible phones
Mobile	7517	26,649
Landline	1901	1616
Premium rate	514	225
Landline or mobile	357	1393
Universal Access Number	180	138
Personal number	131	21
VoIP	125	317
Shared cost	62	158
Pager	55	55
Toll free	30	1418
Voicemail	0	21

**Origin of phone numbers.** We analyzed the numbers to which calls were attempted in our system. As we discussed in Section 3, we found a total of 10,872 valid numbers (for both strong and weak credentials) and 35,011 possible ones. We plot the geographical information of the valid phone numbers on a map to depict where the calls that were attempted were directed. Fig. 8 shows a heat map of the different phone numbers found regardless of whether they were called with strong or weak credentials. The numbers can be mapped to the region to which they belong, and not the exact location of the call. We can see from the heat map that most of the unique phone numbers belong mainly to Europe, Africa, Russia and the Havana and Cuba area. However, the most frequently called countries were the United Kingdom, the United States, Russia and France with 3028, 1063, 435 and 428 calls, respectively.

**Types of phones.** The type of phone numbers which the attackers tried to call was extracted. We used the phonenumbers library to obtain the type of phone number based on the patterns that each country uses for its phone numbers. In some regions, there is no distinction between landline and mobile phone numbers (e.g. the USA), and these cases are included in the landline or mobile numbers category. Table 5 shows the different phone numbers found that were either valid or possible. It can be seen that most of them belong to mobiles or landlines. It is important to note that the possible numbers are local ones that could be valid in some region and, therefore, are not completely accurate since some regions have similar dialing plans and it might be the case, that this number is of another type depending on the region to which it belongs.

The rest of the phone numbers are as follows:

- Premium rate. Telephone numbers through which different services are offered and whose call prices are higher than a conventional call.
- UAN. This is a phone number that allows a company to have several lines associated with it. In this way, when the UAN number is called, one of these lines will be reached depend-

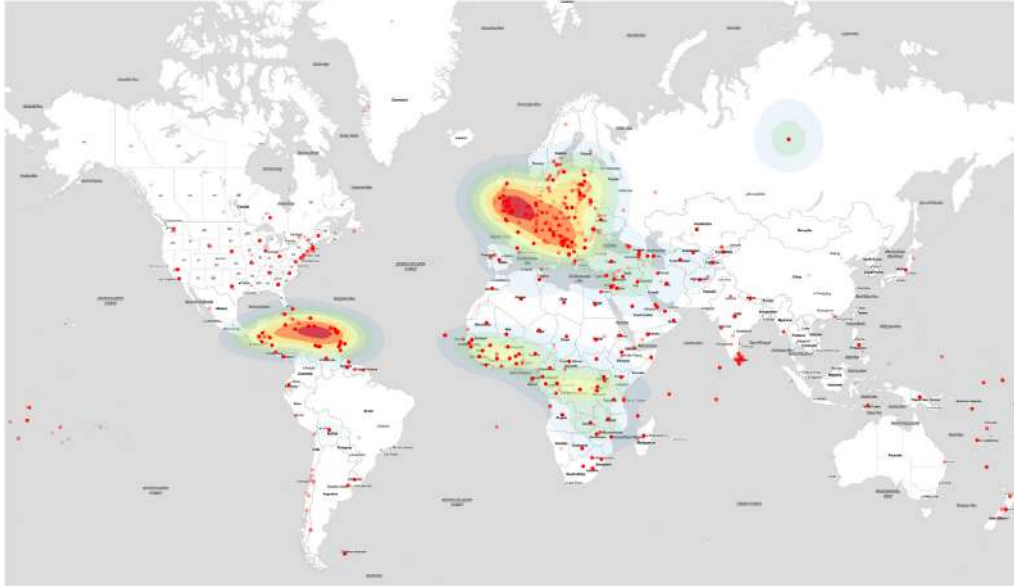


Fig. 8. Geographical map of the location of the phones.

ing on, for example, the geographical location or the time of the call.

- Personal number. This corresponds to virtual phone numbers belonging to the United Kingdom. These phone numbers allow the routing of phone call to another number and have been reported as being used for fraudulent purposes such as fraud or spam [Ofcom \(2016\)](#).
- VoIP. Phone numbers that are assigned to a user instead of a specific phone line. These are contracted through VoIP service providers.
- Shared cost. Telephone numbers for which the cost of the call is divided between the owner of the number and the caller.
- Pager. Phone numbers assigned to devices that are capable of receiving and displaying messages, also known as "beepers" ([Wikipedia, 2022](#)).
- Toll free. These are free phone numbers for the person making the call, as the cost is billed to the owner of the phone number.

The rest of the telephone numbers that attackers attempted to call are invalid, that is, the size or dialplan does not match any region. However, most of these numbers do consist of valid numbers that have been concatenated with numbers or symbols at the beginning. For example, the telephone number "0000000000\*\*4412XXXXXXX" is not valid, but it can be seen that, if the initial part is omitted, the number that remains is a valid number<sup>2</sup> with country code 44, which is the code for the United Kingdom. This pattern is repeated for all the invalid numbers with different digits, symbols and size. We believe it may be a way to try to evade the established dialplan or to know whether they are in a honeypot. For example, if calls to numbers that are not valid give a ringtone, it may indicate that the system is a honeypot.

<sup>2</sup> We have replaced part of the original number with the letter X so that it cannot be recognized.

*Types of fraud.* As we mentioned in [Section 2.3](#), we rely on the taxonomy described in [Sahin et al. \(2017\)](#) to identify the different fraud schemes that were attempted through our system. This taxonomy identifies the different weaknesses and techniques that lead to the different types of fraud and the benefits that they can generate. First, we look at the different fraud schemes identified and the possible benefits attackers can obtain through them. Then, we examine the weaknesses and techniques that allow fraudsters to carry out such actions. The fraud schemes detected are the following:

- Toll Evasion Fraud. This type of fraud is the best known and most widely used over recent years in telephone networks. It is based on making calls without having to pay the charges of the call, which are billed to another party. In this specific case, when accessing the user accounts of a PBX to call, it is the owner of the PBX or the user of the compromised account that is charged for the calls. The benefits can range from simply not paying for anonymous calls involving criminal activities to committing other types of fraud. All the calls that were made were with compromised accounts, so the attackers did not pay for the calls.
- Revenue Share Fraud. This type of fraud occurs when an agreement is made between an operator (or a third-party service provider) and a fraudster in which the latter is responsible for making telephone calls to certain numbers owned by the former (e.g., international premium rate service numbers), generating a revenue which is then shared between them. The benefit of this type of fraud is purely economic, and the more call traffic that is generated, the greater the benefit will be for the fraudsters. As can be seen in [Table 5](#), calls were made to premium numbers, which allow attackers to obtain benefits through these special rate numbers. Also, it can be seen in [Fig. 8](#) that a large part of the telephone numbers belong to the area of Havana and Cuba. These numbers maintain a dialplan



similar to that of the United States, although many correspond to special tariffs.

- Voice Spam and Scams. This scheme includes any type of unwanted or abusive calls, such as spam calls or scams. These calls can be made in many ways, from a phishing call posing as a company to telemarketing and robocalling with prerecorded messages. The possible benefits that can be obtained through this type of fraud include collecting information about the users (e.g., whether the phone belongs to a real user), and convincing or deceiving the recipients of the call to carry out some action that could also have some economic benefit for the fraudster. Most calls could be classified as this type of fraud, in which attackers could use compromised VoIP systems to launch scam campaigns or to find new potential victims.
- Wangiri fraud. Also known as callback scam or ping call, this is a specific type of voice spam or scam fraud that consists in making a missed call with the aim of making curious users return the call. A key feature of this type of fraud is the use of caller ID spoofing. We can identify this type of scam by observing whether there was any attempt to modify the caller ID and by checking whether the time from the beginning to the end of the call is just a few seconds. The possible benefit that the attackers obtain from this type of fraud is economic, since the telephone number is owned by the scammers and they obtain a benefit from the calls received. About 200 calls tried to change the caller ID (we discarded those that tried to change it to the anonymous caller ID).

The techniques and weaknesses on which these types of attacks are based within the VoIP system are similar. Basically, part of the problem of fraud over mobile phone lines comes from the interconnection of multiple technologies and the existence of a large number of operators and services. There is a lot of variation in the regulation and laws concerning telephony between different countries, which has contributed to an increase over recent years in the amount of telephone fraud. In our experiment, attackers gained access through accounts with weak passwords and account leaks on different paste and forum sites. Although this was done on purpose in order to know what happens when fraudsters have access to a system, it often happens that users with little security awareness use weak passwords and unwittingly allow third parties to access their systems. The techniques available for VoIP aim to compromise a PBX server or user accounts to make calls, even simultaneous ones, in order to increase a fraudster's benefits. Fraud schemes can be related to each other since, in this case, all schemes start with avoiding the cost of the calls that are made. It is important to note that no type of fraud was committed through our honeypot since the calls were directed to an internal extension and were not routed to the telephone network.

## 3.5. Clustering campaigns

In this section, we compute the relationships between the source IP addresses of the calls and the valid telephone numbers receiving the call. We depict the relationships in graph form in Fig. 9. Larger nodes represent source IP addresses, while smaller nodes represent valid phone numbers. The edges connect the IP addresses and phone numbers to which calls were attempted and their thickness represents the number of calls. Black telephone numbers represent those numbers that have a special rate (i.e., international numbers with a premium rate, Montserrat, Barbados, etc.), while black nodes representing the origin of the call indicate that they tried to change the caller id to another phone number (we discard anonymous caller IDs from the representation). The gray clusters represent those IPs that only made one valid call and whose phone number is not related to any other IP addresses.

Now we have grouped the relationships between the source and the destination of the calls, together with the type of fraud, we can analyze the different clusters. This enables us to draw the following conclusions:

- There are different IPs that are related by the phone numbers they call, as some of them match for most of the phone numbers. This indicates that they belong to similar campaigns. For example, the pink and the green clusters are campaigns that target different objectives. The former cluster focuses mainly on the UK, while the latter cluster has a more international objective, including various countries, such as the US, India or Germany. The former campaign mainly leverages IP addresses belonging to known VPNs or proxies, while the latter uses IP addresses that are more scattered and that include residential IP addresses (i.e., bots connecting directly from the home network of the victims).
- We can see campaigns with a particularly characteristic topology. For instance, there are campaigns with IP addresses that only made calls to a single valid telephone number. This happens when there are campaigns that are targeted. We can also see cases where the phone number was used to check whether the credentials work (i.e., leaked credentials in a forum) or attackers checking whether they could make valid calls (i.e., as a previous step before making a brute-force attack).
- Most of the campaigns were motivated by an economic benefit as their main incentive, mostly through the use of special rate or premium numbers. We can also see campaigns launching unwanted calls to phone numbers of common users, namely SPAM calls.
- Only a few campaigns tried to change the caller ID to another phone number, presumably controlled by the attacker, and if so, it would be the Wangiri fraud. Among these campaigns, only two of them share a destination phone number, implying that the other campaigns can be attributed to different actors.

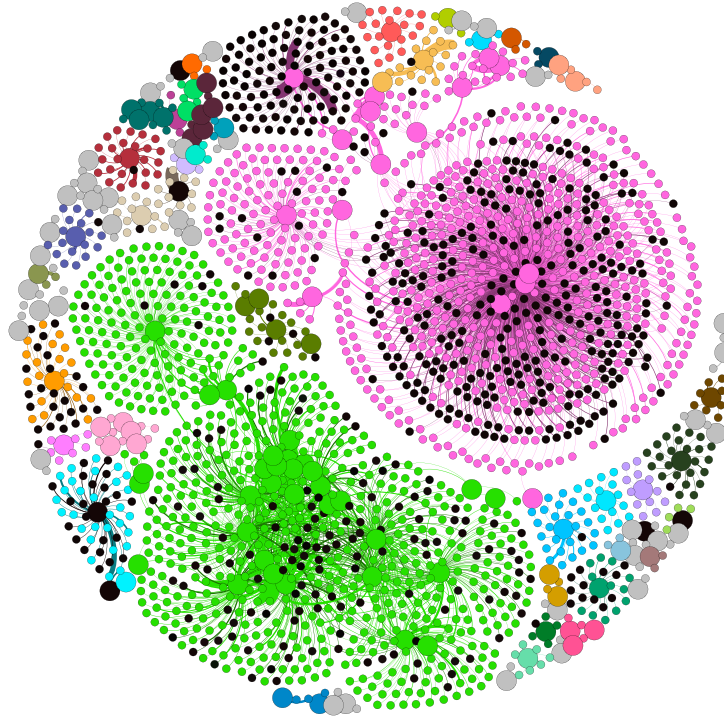
## 3.6. Timestamp analysis

In this section, we provide a timestamp analysis focused on calls to the United Kingdom and the United States, which are the two countries that were called the most (based on the numbers obtained as valid and for which it was possible to obtain data on their geographical location). We use local time for both countries on the basis of the time zone to which the phone number belongs. Fig. 10 shows the graph of calls to each country, distributed both by the times at which the call occurred and the day of the week, regardless of whether they were made in the experiment for weak or strong credentials. By looking at the graphs, we can observe that the time of the calls is targeted at the local customs, that is, they are mainly made during office hours and between 8:00 and midnight, when it is common for people to be at home. Also, it can be seen that, for the UK, a percentage of calls were made between 00:00 and 03:59, which could indicate that certain types of bots make calls once they access the system regardless of the time or day.

Regarding the day of the week, we observe that calls were made on any day and that in the UK the number of calls spikes on Thursday, Friday and Saturday. By contrast, in the United States, the number of calls per day is similar, reaching its peak midweek.

## 4. Discussion

In this section we discuss the limitations of our work, as well as the findings of the study.



**Fig. 9.** Clustering of IP addresses and valid phone numbers. The larger nodes represent the IP addresses from which the calls originated and the smaller nodes represent valid phone numbers. The edges connect IPs and phones if there was a call attempt from the former to the latter.

#### 4.1. Limitations

We encountered a number of limitations in the course of the experiments. On the one hand, due to the ethical requirements that this type of experiments have, calls were not routed to the telephone network; instead, those calls that coincided with the dialplan were directed to a private extension to simulate that calls were routed correctly. Some attackers may have been trying to call a phone number they had control over to check whether the calls were being made correctly. This can cause some of the attackers to realize that they are in a system with certain limitations and thus not to carry out any of the actions that they would perform in a real system. On the other hand, when we carried out credential leaks, these were made on public paste sites and in underground public forums that allow the free registration of any user. For example, in private forums, users rely more on the information provided by other users since an invitation is needed to be part of that community.

**Configuration.** We used the Long Term Support (LTS) 16.3.0 version of asterisk as PBX server, enabling the option “alwaysauthreject = no” so that the server response is different when the identification is wrong in the experiment with weak credentials. That is, it shows a different message when the password is incorrect or when the user does not exist. This makes it easier for attackers to identify users through brute force, making it easier for attackers to find usernames and passwords. User accounts are set to a specific dialplan that allows calls to only certain countries (see

Sections 2.1 and 2.2), trying to simulate a real system where calls to other countries are not allowed. Using other configurations or dialplans and it may reveal other types of attacks by attackers.

#### 4.2. Key findings

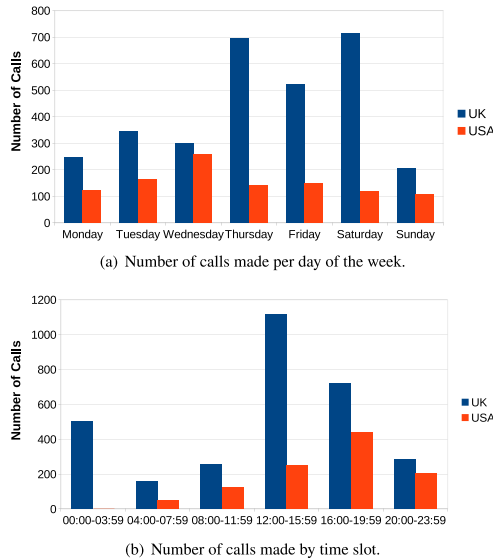
In this work we have shown that VoIP attacks are a threat that underpins part of the malicious call ecosystem. We next summarize the key findings of our work.

**Type of attacks.** We provided for the first time, a study based on a live measurement of the incoming utterances in the calls that shows how fraudsters use a compromised VoIP system. Among a comprehensive list of different frauds we observed, we found that two types stand out in the wild: Toll Evasion and Revenue Share. First, Toll Evasion enables the commoditization of botnets performing telephony fraud in a similar fashion to *pay-per-install* services sold in underground markets (Thomas et al., 2015). Second, we saw how Revenue Share also adopts successful fraudulent schemes such as “affiliate” marketing (McCoy et al., 2012) or “click fraud” (Stone-Gross et al., 2011) in the realm of telephony.

**Depleting.** We can see that scammers try to call certain phone numbers several times. This seems to indicate that they obtain phone numbers through leaked or purchased databases (Tu et al., 2016) and attempt to call them once they have access to a system. Related works have shown that depleting is effective against spam (Stringhini et al., 2012). Using a honeypot may help organizations to keep a list of the phone numbers that scammers call and thus

J. Carrillo-Mondéjar, J.L. Martínez and G. Suarez-Tangil

Computers & Security 119 (2022) 102758



**Fig. 10.** Number of calls that were made to the United States and the United Kingdom. The time zone used is the local time of the geographic location of the phone numbers.

minimize the number of spam calls, which can be very annoying (e.g., when there are call attempts to a number whose caller is trying to make other calls at the same time).

**Source attacks.** Although we know that attackers may use other services to hide their identity, most of the logged IP addresses belonged to Europe. As we saw in Section 3.2, many of them are related and belong to the same organization. Most of these organizations provide hosting services or private and virtual servers for users to set up their own services. Everything seems to indicate that the attackers exploit known vulnerabilities in services hosted on these machines as the software has often not been properly patched. Although nowadays many resources and efforts are invested in raising awareness among users regarding security issues and keeping their systems up to date, there are still many vulnerable devices accessible from the Internet that have not received the necessary patches.

**Calls.** We observed that most of the calls that were directed to the United States and the United Kingdom were made within the time zone corresponding to each country, thus increasing the chances of their success. However, some calls fell outside the usual time slot, indicating that some bots make calls without assessing the context of the target. In addition, call patterns were found to generate numbers that are invalid. As we saw in Section 3.4, these numbers are made up of valid numbers to which prefixes of different sizes have been added with numbers or symbols. This indicates that the bots are trying to actively bypass the restrictions of the dialplan used by the VoIP server. Introducing prefixes can cause errors in the mechanism used to enforce restrictions, which in turn can let attackers evade these mechanisms and make calls. This behavior, however, is very noisy and can be leveraged to design a system to systematically detect these bots. We can also see that mobile and landline calls are the most targeted type of services as opposed to premium rate services, as was initially expected. This

makes it challenging to deploy mitigating actions that rely on the dialplan being used.

**Accounts.** Fraudsters actively search for user accounts, scanning the Internet for known VoIP server ports to which they try to connect and apply brute-force login credentials. In addition, they also make use of leaked accounts on different sites, and it is more than likely that sales of stolen accounts take place. Although with strong credentials far fewer calls were made compared with accounts with weak credentials, attackers did access the former and similar behaviors were observed.

## 4.3. Countermeasures

In this section we describe a series of countermeasures to fortify VoIP servers and thus prevent a fraudulent use being made of them.

**Accounts.** As we have seen, attackers scan VoIP servers and make connection attempts by sweeping users and passwords in order to find weak credentials that allow them to gain access. Depending on the server configuration, it is possible to enumerate valid users based on the response from the server. In the authentication process, the server can respond with the message "Not Found" or "Unauthorized" (Jansky et al., 2017) depending on whether the user is correct or not. A good practice is to configure the server so that the response is the same regardless of whether the user exists but the password is incorrect or the user does not exist. Although the default configuration of Asterisk in the latest versions avoids giving different messages in the authentication process, it can be modified or configured erroneously by the server administrator in the config files. Also, on the basis of the brute force attacks received, we can see that most of them attack numeric users, so using named users would make the task of finding users more difficult for the attackers. Finally, in the event of an attacker obtaining valid credentials, either by brute force or by theft, for example by using malware on mobile devices, a good measure is to continuously monitor the calls made and limit the number of simultaneous calls that can be made through the server. Usually, once they access the server, the attackers try to make numerous simultaneous calls, which can cause great economic losses (Osenbaugh, 2019).

**Dialplan.** Fraudsters try to make calls to phone numbers in different parts of the world, including attempts to call strange phone numbers in order to avoid the restrictions of the dialplan configured for the user. It is important to implement a correct dialplan configuration and only allow calls to places where you operate or where you usually make calls, including implicit rules for the types of number you want to allow and disable calls to international and premium numbers.

**Architecture.** To deal with attacks against SIP servers, a good measure is to implement an SIP proxy server (e.g., (Kamailio and Kamailio, 2022)). This allows you to load balance and redirect valid packets to the corresponding SIP server and release the load from the SIP server. Through a proxy, security is increased since it makes it possible to discard and ban brute-force attacks, ban the default user agent of known tools (e.g., Sipvicious (Gauci, 2022) or Sipscan (Endler and Collier, 2022)), provide the same error message in the authentication process<sup>3</sup> regardless of whether the username or password is incorrect and implement other security measures. It is also possible to ban brute-force attempts on the server itself via iptables with solutions such as Fail2ban (Jaquier, 2022)). Another good measure is to change the default port of the SIP server to avoid being found by automatic scanners looking for the default VoIP ports.

<sup>3</sup> See limitation of dictionary attacks with the HTable module in Kamailio at: <https://kamailio.org/docs/modules/stable/modules/htable.html#idm49>.

## 5. Related work

In this section we compare this paper with previous related works.

**Honeypot systems.** Nawrocki et al. (2016) review the state of the art of the different honeypot software systems and the data they collect by making a classification based on the type of interaction that the attackers have as well as the services and applications they emulate. Luo et al. (2017) propose a honeypot system for different Internet of Things (IoT) architectures with the aim of detecting attacks or even zero days at their earliest stage. Similar work is carried out in Pa et al. (2015), in which support is provided for eight IoT architectures and connections based on the Telnet protocol. Unlike previous works, Dowling et al. (2017) propose a method for Wireless Personal Area Networks (WPANs) which enables the detection of different types of attack in the Zigbee protocol, which is part of WPAN. These types of honeypot mainly are mainly focused on detecting attacks that occur on their systems in order to detect new threats or types of attack aimed at compromising a system or violating the privacy of users. Our approach is a real VoIP system, configured to limit attackers and focused on obtaining knowledge through the attacks and actions carried out through it.

**Honeypot accounts.** Kaur et al. (2018) carry out a review of all the related works on the different approaches designed to combat spam and compromised accounts in social networks by analyzing each of the previous studies and discussing their pros and cons. Bursztein et al. (2014) study the manual hijacking of accounts with a focus on emails and phishing websites intended for Google users. The study focuses on how user credentials are captured by attackers and used once they access the user's account. Onalapo et al. (2016) carry out a similar investigation with the difference that, instead of focusing on phishing attacks, they use a broader threat model by looking at user credentials in Gmail that have been automatically stolen by malware. They also analyze the behavior of cybercriminals when they gain access through leaked credentials on paste sites and forums. Other studies have focused on the study of user accounts that were under the control of spammers in different social networks. Thomas et al. (2011) investigate the abuse by spammers of the social network Twitter. To do this, they collect a dataset of all tweets from the accounts that were suspended by the tweeter to characterize the behavior of spammers in the social network. Stringhini et al. (2010) analyze the behavior of spammers in social networks using 300 honey accounts in three of the main social networks in order to develop techniques that allow detecting spammers. Unlike our work, they focus on identifying the behavior of user accounts belonging to social networks and email instead of the behavior of attackers in compromised VoIP accounts. In our study, we use "real" accounts that are obtained by attackers either through brute force attacks or from different forums or paste sites, aiming to give the impression that the accounts have been stolen.

**Telephone spam and scams.** Tu et al. (2016) reviewed the state of the art of the techniques used against telephone scams. They describe the ecosystem of spam calls focusing on the differences with spam in emails. Finally, they analyze the different existing solutions against telephone spam. Miramirkhani et al. (2016) propose the first study of the technical support of the scams and the call centers that are behind them. They build an automatic system to detect phone numbers and domains that are used by scammers. Finally, they make calls to the numbers of the scammers to interact with them and collect statistical details of the techniques and the process used by them. Li et al. (2018) design a set of features to feed machine learning algorithms in order to detect spam calls and scams. As they have no access to mobile telephony infrastructures, they develop a mobile application for users to la-

bel calls as malicious and thus enable them to build a dataset to analyze and extract features. Sahin et al. (2017) use a chatbot to connect unwanted calls to the bot, which mimics a person's behavior. Although the chatbot used is not based on any advanced artificial intelligence, it proves very effective, managing to keep calls for 10 min, causing spammers to waste time and resources interacting with a bot. Tu et al. (2019) conduct a study to understand why users fall into traps and end up being victims of scams. To do this, they conduct 10 telephone phishing experiments on 3000 university students. Among all the possible factors that could exist to carry out the deception successfully, they observe that the impersonation of the caller's ID prevailed in most cases. Bullée et al. (2016) conduct a study on the awareness of workers regarding the social engineering attacks they may receive, in this case over the telephone. According to the results obtained, they verify that scam awareness campaigns reduce user exposure only for a short period of time. Prasad et al. (2020) propose a method to classify calls based on the analysis of the audio and metadata of the calls, which allows them to cluster different types of audios and fraud campaigns. In general, previous work has focused on detecting scam calls when they are received and interacting with scammers either by directly calling the phone numbers recorded via the Internet or with a chatbot when receiving calls. On the contrary, our work is focused on how scammers obtain infrastructure or resources to make calls at no cost and what type of actions they perform by monitoring a real VoIP system, thus providing insights into what types of fraud are currently being carried out through compromised VoIP systems.

## 6. Conclusions

Telephony fraud has been a long-standing and challenging problem since the invention of the telephone. Unfortunately, with the emergence of VoIP, fraudsters now have access to a more powerful technology which allows them to carry out more sophisticated and complex attacks and to target an immense number of victims.

In this paper, we have presented a study of the behavior of criminals when they acquire VoIP user accounts, either through the use of brute force or by using accounts leaked via paste sites or through underground markets. In order to do so, we built a VoIP honeypot and established two experiments: one using accounts with weak credentials, and the other with strong credentials that were previously leaked. Then, we studied how scammers used these compromised accounts and provided a holistic overview of their activity, including provenance and intent. The different types of fraud that were attempted through our systems were identified, as well as the techniques used to increase the possible benefit that the fraudsters can obtain through these attacks.

We identify both Toll Evasion and Revenue Share as being two of the most prevalent types of fraud. This finding can be used to inform law enforcement as well as the public to deploy educational countermeasures. Furthermore, we contextualized fraudulent calls and studied key patterns such as: the time, the location and the hosting used by the fraudsters when connecting to our system. We can see that this context is valuable when deploying technical countermeasures (e.g., depleting fraudsters' resources). The results show that more than 50% of the fraudsters that interacted with the system did so through IP addresses in Europe, with the Netherlands making the highest number of outgoing calls. The UK was the most called country on the basis of the valid phone numbers that were dialed, followed by the USA. Furthermore, it was noted that a large number of the IP addresses detected belonged to hosting services, which seems to indicate that these sites were compromised by fraudsters and used to hide their identity. This assumption is also supported by the fact that, after analyzing these

IP addresses, we found that they had multiple commonly opened ports, as well as others that are less ordinary, such as the one used for the RDP protocol, which is known to have a high criticality vulnerability. In fact, the addresses were scanned for vulnerabilities, obtaining results indicating that most of them were of medium and high criticality.

With this work, we have gained a better understanding of the actions performed by attackers when they have access to a VoIP server. We have also presented a number of countermeasures to tackle this threat. We believe that our findings are key to deploying effective mitigating actions.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## CRediT authorship contribution statement

**J. Carrillo-Mondéjar:** Conceptualization, Software, Validation, Data curation, Investigation, Writing – original draft, Visualization, Writing – review & editing. **J.L. Martínez:** Conceptualization, Methodology, Investigation, Writing – original draft, Writing – review & editing, Supervision, Project administration, Funding acquisition. **G. Suarez-Tangil:** Conceptualization, Methodology, Investigation, Resources, Writing – original draft, Writing – review & editing, Supervision.

## Acknowledgments

This work has been supported by the Ministry of Economic Affairs and Digital Transformation, Spain under project RTI2018-098156-B-C52, by the Regional Government of Castilla-La Mancha under the project SBPLY/17/180501/000353 and SB-PLY/21/180501/000195, by the Spanish Education, Culture and Sports Ministry under grant FPU 17/03105, and by the “Ramon y Cajal” Fellowship RYC-2020-029401-I.

## References

- Anabo, I.F., Elexpuru-Albizuri, I., Villardón-Gallego, L., 2019. Revisiting the belmont report's ethical principles in internet-mediated research: perspectives from disciplinary associations in the social sciences. *Ethics Inf. Technol.* 21 (2), 137–149. doi:10.1007/s10676-018-9495-z.
- Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., Zhou, Y., 2017. Understanding the mirai botnet. In: 26th USENIX Security Symposium (USENIX Security 17). USENIX Association, Vancouver, BC, pp. 1093–1110. <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>
- Bullée, J.-W., Montoya, L., Junger, M., Hartel, P.H., 2016. Telephone-based social engineering attacks: an experiment testing the success and time decay of an intervention. In: SG-CRC, pp. 107–114.
- Bursztein, E., Benko, B., Margolis, D., Pietraszek, T., Archer, A., Aquino, A., Pitsillidis, A., Savage, S., 2014. Handcrafted fraud and extortion: manual account hijacking in the wild. In: Proceedings of the 2014 Conference on Internet Measurement Conference - IMC '14. ACM Press, Vancouver, BC, Canada, pp. 347–358. doi:10.1145/2663716.2663749.
- Chronicle, Virustotal. <https://www.virustotal.com/>.
- Cvedetails, CVE security vulnerability database. <https://www.cvedetails.com/>.
- Dittrich, D., Kenneally, E., et al., 2012. The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research. Technical Report, US Department of Homeland Security.
- Dowling, S., Schukat, M., Melvin, H., 2017. A ZigBee honeypot to assess IoT cyber-attack behaviour. In: 2017 28th Irish Signals and Systems Conference (ISSC), pp. 1–6. doi:10.1109/ISSC.2017.7983603.
- Drysdale, D. Phonenumbers: Python version of Google's common library for parsing, formatting, storing and validating international phone numbers. <https://github.com/daviddrysdale/python-phonenumbers>.
- Endler, D., Collier, M. Hacking tools: sipsan. <https://en.kali.tools/all/?tool=1253>.

- Getipintel, Free Proxy / VPN / TOR / Bad IP Detection Service via API and Web Interface | IP Intelligence. <https://getipintel.net/>.
- Gauci, S. Kali tools: Sipsan package description.
- Jansky, T., Čejka, T., Bartoš, V., 2017. Hunting sip authentication attacks efficiently. In: Tuncer, D., Koch, R., Badonnel, R., Stiller, B. (Eds.), *Security of Networks and Services in an All-Connected World*. Springer International Publishing, Cham, pp. 125–130.
- Jaquier, C. Fail2ban. <https://www.fail2ban.org/>.
- Kambourakis, G., Kolias, C., Stavrou, A., 2017. The mirai botnet and the IoT zombie armies. In: MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM), pp. 267–272. doi:10.1109/MILCOM.2017.8170867. ISSN: 2155-7586.
- Kaur, R., Singh, S., Kumar, H., 2018. Rise of spam and compromised accounts in online social networks: a state-of-the-art review of different combating approaches. *J. Netw. Comput. Appl.* 112, 53–88. doi:10.1016/j.jnca.2018.03.015.
- Klimt, B., Yang, Y., 2004. Introducing the enron corpus. CEAS.
- Li, H., Xu, X., Liu, C., Ren, T., Wu, K., Cao, X., Zhang, W., Yu, Y., Song, D., 2018. A machine learning approach to prevent malicious calls over telephony networks. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 53–69. doi:10.1109/SP.2018.00034.
- McCoy, D., Pitsillidis, A., Grant, J., Weaver, N., Kreibich, C., Krebs, B., Voelker, G., Savage, S., Levchenko, K., 2012. Pharmaleaks: understanding the business of on-line pharmaceutical affiliate programs. In: 21st (USENIX) Security Symposium ((USENIX) Security 12), pp. 1–16.
- Luo, T., Xu, X., Jin, X., Jia, Y., Ouyang, X. IoT CandyJar: towards an intelligent interaction honeypot for IoT Devices, Black Hat, 2017.
- Microsoft, 2019. CVE-2019-0708 | Remote Desktop Services Remote Code Execution Vulnerability. <https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2019-0708>.
- Mitre corporation, T., 2019. CVE common vulnerabilities and exposures. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-0708>.
- Miessler, D., Haddix, J. g0tm1k, danielmiessler/SecLists. <https://github.com/danielmiessler/SecLists>.
- Miracle, V.A., 2016. The belmont report: the triple crown of research ethics. *Dimens. Crit. Care Nurs.* 35 (4), 223–228.
- Miramirkhani, N., Starov, O., Nikiforakis, N., 2016. Dial one for scam: analyzing and detecting technical support scams. *CoRR abs/1607.06891* 1607.06891. <http://arxiv.org/abs/1607.06891>.
- Navrocki, M., Wählich, M., Schmidt, T. C., Keil, C., Schönfelder, J., 2016. A survey on honeypot software and data analysis. *CoRR abs/1608.06249* <http://arxiv.org/abs/1608.06249>.
- Ofcom, Consumers warned about 070 missed call scam.
- Onalapo, J., Mariconti, E., Stringhini, G., 2016. What happens after you are pwnd: understanding the use of leaked webmail credentials in the wild. In: Proceedings of the 2016 Internet Measurement Conference. ACM, New York, NY, USA, pp. 65–79. doi:10.1145/2987443.2987475. Event-place: Santa Monica, California, USA.
- Osenbaugh, J. Telecom fraud on the rise: what enterprises need to know.
- Pa, Y.M.P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., Rossow, C., 2015. Iotpot: analysing the rise of IoT compromises. 9th USENIX Workshop on Offensive Technologies (WOOT 15). USENIX Association, Washington, D.C. <https://www.usenix.org/conference/woot15/workshop-program/presentation/pa>
- Prasad, S., Bouma-Sims, E., Mylappan, A.K., Reeves, B., 2020. Who's calling? Characterizing robocalls through audio and metadata analysis. In: 29th USENIX Security Symposium (USENIX Security 20). USENIX Association, pp. 397–414. <https://www.usenix.org/conference/usenixsecurity20/presentation/prasad>
- Proffitt, C., Wolf, D. Bluekeep:Havoc on the horizon.
- Project, T. K. S. S. Kamailio sip server. <https://www.kamailio.org/w/>.
- Sahin, M., Francillon, A., Gupta, P., Ahamad, M., 2017. Sok: fraud in telephony networks. In: 2017 IEEE European Symposium on Security and Privacy (EuroSP), pp. 235–250.
- Sahin, M., Relieu, M., Francillon, A., 2017. Using chatbots against voice spam: analyzing lenny's effectiveness. In: Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017). USENIX Association, Santa Clara, CA, pp. 319–337. <https://www.usenix.org/conference/soups2017/technical-sessions/presentation/sahin>
- Sangoma Technologies, 2021. Open Source Communications Software | Asterisk Official Site. <https://www.asterisk.org/home>.
- Shodan, Shodan is the world's first search engine for internet-connected devices. <https://www.shodan.io/>.
- s.r.o. C. H. Multiple vulnerabilities in microsoft windows SMB server. <https://www.cybersecurity-help.cz/vdb/SB2017031416>.
- Stone-Gross, B., Stevens, R., Zarras, A., Kemmerer, R., Kruegel, C., Vigna, G., 2011. Understanding fraudulent activities in online ad exchanges. In: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference, pp. 279–294.
- Strand, L., Leister, W., 2011. Improving SIP authentication.
- Stringhini, G., Egele, M., Zarras, A., Holz, T., Kruegel, C., Vigna, G., 2012. B@bel: leveraging email delivery for spam mitigation. In: Presented as Part of the 21st (USENIX) Security Symposium ((USENIX) Security 12), pp. 16–32.
- Stringhini, G., Kruegel, C., Vigna, G., 2010. Detecting spammers on Social Networks. In: Proceedings of the 26th Annual Computer Security Applications Conference. ACM, New York, NY, USA, pp. 1–9. doi:10.1145/1920261.1920263. Event-place: Austin, Texas, USA.
- Thomas, K., Grier, C., Song, D., Paxson, V., 2011. Suspended accounts in retrospect: an analysis of twitter spam. In: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference. ACM, New York, NY, USA, pp. 243–258. doi:10.1145/2068816.2068840. Event-place: Berlin, Germany.



- Thomas, K., Huang, D., Wang, D., Bursztein, E., Grier, C., Holt, T.J., Kruegel, C., McCoy, D., Savage, S., Vigna, G., 2015. Framing dependencies introduced by underground commoditization. *Workshop on the Economics of Information Security*.
- Tu, H., Doupé, A., Zhao, Z., Ahn, G., 2016. Sok: everyone hates robocalls: a survey of techniques against telephone spam. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 320–338. doi:10.1109/SP.2016.27.
- Tu, H., Doupé, A., Zhao, Z., Ahn, G.-J., 2019. Users really do answer telephone scams. In: 28th USENIX Security Symposium (USENIX Security 19). USENIX Association, Santa Clara, CA, pp. 1327–1340. <https://www.usenix.org/conference/usenixsecurity19/presentation/tu>
- Vetterl, A., Clayton, R., 2018. Bitter harvest: systematically fingerprinting low- and medium-interaction honeypots at internet scale. 12th USENIX Workshop on Offensive Technologies (WOOT 18). USENIX Association, Baltimore, MD. <https://www.usenix.org/conference/woot18/presentation/vetterl>
- Wikipedia. Pager. <https://en.wikipedia.org/wiki/Pager>.



Jyväsylä.

**Javier Carrillo-Mondéjar** received a B.Sc. degree in Computer Science and a Master's degree in Advanced Computer Science from the University of Castilla-La Mancha, Spain, in 2016 and 2017, respectively. Currently, he is enrolled full-time on the Ph.D. Program in Advanced Information Technology at this university. In 2016, he joined the Computer Architecture and Technology Group of the Informatics Research Institute of Albacete (I3A) as a researcher. His research interests are related to malware detection and classification techniques, as well as the methods used in malware to spread and remain hidden in computer systems. He has also been a visiting researcher at King's College London and the University of



and transcoding, and topics related to security. He has also been a visiting researcher at the Florida Atlantic University, Boca Raton (USA), and the Centre for Communication System Research (CCSR), at the University of Surrey, Guildford (UK). He has over 100 publications in these areas in international refereed journals and conference proceedings.



**Guillermo Suarez-Tangil** is with IMDEA Networks Institute. His research focuses on systems security and malware analysis and detection. In particular, his area of expertise lies in the study of smart malware, ranging from the detection of advanced obfuscated malware to the automated analysis of targeted malware. Previously, he was Lecturer (Assistant Professor) at King's College London (KCL). Before joining KCL, he was senior research associate at University College London (UCL), where he was also actively involved in other research areas involved with detecting and preventing Mass-Marketing Fraud.

## CHAPTER 4

### A chatbot for selectively engaging with online fraudsters

- **Title:** A chatbot for selectively engaging with online fraudsters.
- **Authors:** J. Carrillo-Mondéjar, J.L Martínez and G. Suarez-Tangil.
- **Type:** Journal paper.
- **Journal:** IEEE Access.
- **Publisher:** IEEE.
- **ISSN:** 2169-3536.
- **Status:** Under review.
- **JCR IF/ranking:** 3.476/Q2 (JCR2021).

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

# A chatbot for selectively engaging with online fraudsters

J. CARRILLO-MONDÉJAR<sup>1</sup>, J.L. MARTÍNEZ<sup>1</sup> AND G. SUAREZ-TANGIL<sup>2</sup>

<sup>1</sup>University of Castilla-La Mancha, Albacete, Spain (e-mail: [javier.carrillo.joseluis.martinez]@uclm.es)

<sup>2</sup>IMDEA Networks Institute, Madrid, Spain (e-mail: [guillermo.suarez-tangil]@imdea.org)

Corresponding author: J. Carrillo-Mondéjar (e-mail: javier.carrillo@uclm.es).

This work has been supported by the MINECO and European Commission (FEDER funds) under project RTI2018-098156-B-C52, by the JCCM under the projects SBPLY/17/180501/000353 and SBPLY/21/180501/000195, by the Spanish Education, Culture and Sports Ministry under grant FPU 17/03105, and by the "Ramon y Cajal" Fellowship RYC-2020-029401-I.

**ABSTRACT** Telephone scams are a longstanding problem that has proliferated significantly with the advent of Voice over IP (VoIP) technology. VoIP allows scammers to establish low-cost call centers in remote countries with lax anti-fraud regulations, reaching victims worldwide through local virtual numbers. This type of scam causes significant financial damage to both individuals and companies. Approaches that deal with this threat rely primarily in call headers (e.g., Caller-ID), which are easily evaded (e.g., through spoofing). In this paper, we instead leverage the actual content of the calls to mitigate telephony fraud. Our work presents the first end-to-end system for automatically detecting and engaging with telephony scammers online. Our system first builds a machine learning-based model to identify and classify incoming calls. We then develop two chatbots able to analyze conversations and offer meaningful responses in real-time. The first chatbot uses pre-recorded utterances of a wide-range of responses in a closed-set read out by a native speaker. The chatbot is inherently limited by a pre-determined set of rules for maintaining a conversation, for which we use the Rasa Natural Language Understanding (NLU). The second chatbot generates responses in an open set using a Generative Pre-trained Transformer 3 (GPT-3) network and speech synthesis. This let us deliver responses to the attacker tailored to the context of the conversation, but the utterances are inherently more artificial. We evaluate both solutions by setting an IRB-approved experiment in which the chatbots are tested with real conversations with scammers, and we measure their ability to engage in conversation and maintain appropriate-length calls.

**INDEX TERMS** Chatbot, Honeypots, Machine Learning, Robocalls, Telephony fraud, Underground economy, VoIP

## I. INTRODUCTION

UNWANTED or malicious phone calls, known as telephone spam and scams, have been a longstanding and challenging problem. In 2021 alone, this type of fraud has generated losses of billions of dollars worldwide [1], [2] — not to mention the annoyance they produce on victims. Unlike traditional spam or scam email, phone calls are more intrusive and they are difficult to avoid or to block a priori. Different types of offenses can be carried out through telephone lines, ranging from abusive advertising calls to telephone scams or extortion. One example is the tech support scam, in which the scammer uses deceptive web pop-ups to convince users that their machine is infected and encourages them to call to a given tech support center to fix the problem. Other more invasive examples are calls to telephone users in which the fraudster impersonates the identity of a well-

known technological company (e.g., Microsoft), indicating that they have detected that their computer is infected, luring the victims to give the scammers the control of the machine.

The proliferation of VoIP has fostered the telephone fraud ecosystem, as it allows cybercriminals to reach a larger number of victims while reducing the operational costs and complexity in their campaigns. It also lets the scammers run their operations in countries with less restrictive regulations. Malicious calls are typically carried out in the form of robocalls, which are dispatched automatically via software that uses dialing equipment to generate calls to sets of numbers, either given on a list or chosen randomly. When a victim answers the call, the robocall plays a pre-recorded message. In other cases, once the call has been answered, it is transferred to a live agent that handles the scam. This is however a limiting factor for fraudsters as live agents are not always



available to answer the call. In either case, robocalls make it possible to reach a large number of victims in a short period of time.

Despite the magnitude of this type of cybercrime and its proliferation, there is an absence of academic literature on practical methods for detecting telephone fraud. Countermeasures commonly used to filter email spam can not be easily adapted to prevent telephone systems [3]. This is mainly because telephone scams are not generally run in large-scale campaigns, and except for robocalls they are not generated automatically. As a result, they cannot be identified by the similarity-detection methods used for spam filtering similar to other hand-crafted scams [4]. Prior work addresses the problem through analyzing the call headers [5] (e.g., Caller-ID blacklist, Caller-ID Anomaly, or using Reputation System). However, these types of measurements can be bypassed by spoofing the Caller-ID [3]. Other related work focus on dynamic approaches, which are based on the analysis of the audio of the call. Authors in [6] detect whether the call contains interactive voice responses, while the authors of [7] analyze the audio footprints to detect whether it is similar to previous campaigns. The work in [8] studies the feasibility and effectiveness of an interactive response system called Lenny as a measure to combat scams. One of the most noteworthy aspects of their work is the use of chatbots to waste fraudsters' time and thus reduce the resources and time they can spend on other potential victims (namely scambaiting).

In this paper, we build a chatbot architecture for interacting with scammers in real time. Our system address the challenge of telephone fraud through a multi-pronged analysis of the types of telephone fraud. Using this analysis, we build a machine-learning classifier that can distinguish across the most common types of scams. We then build an system than can automatically engage into an autonomous conversation with the scammer using chatbots. While the objective of the classifier is to detect telephone scams, the chatbots aim at depleting the resources of the scammer. Below we formulate five research questions in order to assist our analysis and the evaluation of our system:

- **(RQ1)** Are engagement and mitigating strategies more effective when they are tailored to the type of telephony scam? Is our system capable of systematically classifying the type of unwanted call?
- **(RQ2)** What are the main terms that should alarm users in a telephone conversation?
- **(RQ3)** What strategies work best when interacting with scammers and what challenges arise from the type of scam?
- **(RQ4)** Is our system capable of interacting with fraudsters and wasting their time by generating appropriate responses in real time?
- **(RQ5)** What are the key features that contribute to the engagement of the scammer? Do more realistic (in terms of speech) but less smart (in terms of richness in the response) chatbots outperform less realistic but smarter chatbots?

Our main contributions can be summarized as follow:

- We develop a system architecture to mitigate unwanted calls that works autonomously and in real-time to cope with the problem of telephone fraud at scale.
- We build a spam and scam classifier, following a Knowledge Discovery from Data (KDD) approach [9]. We evaluate its performance using different metrics and classification algorithms.
- We design two different chatbot strategies and compare their performance. The first one is based on Rasa NLU, which responds with pre-recorded answers. The second one is based on the Generative Pre-trained Transformer 3 (GPT-3). The model has been trained using deep learning models designed for generating answers within the context of a conversation. These models allow us to generate conversations with scammers in real time.
- We evaluate the effectiveness and viability of our system with real calls to scammers.

The rest of the paper is organized as follows. First, in Section II we discuss the threat posed by scam phone calls, then we detail the architecture of our system in Section III. Section IV describes the ethical principles followed in the experiments. Section V shows the results of the spam and scam call classifier. Section VI presents a system evaluation with real-time conversations. Section VII discusses the proposals from the community related to our work. Finally, the main conclusions of this work are presented in Section VIII.

## II. THREAT MODEL AND OVERVIEW

The use of VoIP networks is nowadays increasing and, in many cases, replacing public telephone interconnection infrastructures (e.g., companies, organizations or houses). Attacks leveraging this type of network have recently become more and more common [10]. Due to the interconnection of VoIP systems with traditional telephone network infrastructures, attackers have the ability to target a greater number of victims around the world.

### A. THREAT MODEL

We identify two scenarios through the analysis of the thread model that drives the illicit usage of VoIP.

**Infrastructure:** We see attacks to VoIP systems aiming at stealing access credentials. Once attackers take control a set of accounts, they use them place calls without paying for their cost. In some cases, attackers target VoIP servers themselves, effectively obtaining root access and full control of the infrastructure. This way they can issue a large volume of calls and maximize the profit obtained in the different types of phone fraud they carry (e.g., premium rate services) [11]. Controlling the entire VoIP server also give the attackers access to the list of active numbers, which enables them to launch other types of attacks (e.g., Wangiri scam) [12]. This causes considerable economic losses for the owners of the VoIP infrastructure or the credentials used [1], [2].

**Call centers:** On the other hand, the availability of local inexpensive dial-up plans has turn VoIP networks into a

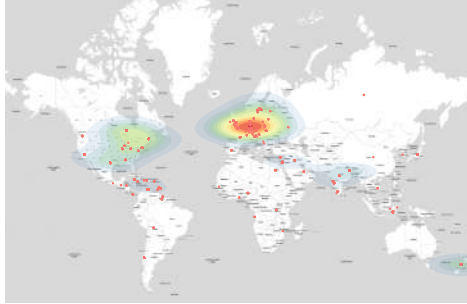


FIGURE 1. Heat map of the destinations to which calls were attempted.

commodity for call centers. This makes it technical viable for cybercriminals to launch phone scam campaigns. The aim of these campaigns is to trick legitimate users into carrying out certain actions that carry a financial loss in their end.

### B. CHARACTERIZING THREATS

To better understand telephone scams, we configure a honeypot for a VoIP system. The honeypot runs a VoIP server with weak credentials over an IP address publicly accessible from the Internet. Our aim is to measure the attacks that these networks receive, but without actually redirecting the phone calls out of our honeypot. Figure 1 shows a heat map with the calls that were attempted through the honeypot. In total, 92,605 calls were attempted, 2,955 of which were to valid numbers or ones that contain information on the country or area to which they belong. It can be seen that most of the attempted calls were to places in Europe and the United States. We can also see that attempts were made to make calls to phones with premium rates and whose call plan is similar to that of the United States (e.g., Dominican Republic, Cuba). In addition, Figure 2 shows a graph of the set of IP addresses that interacted with our system, grouping the IP addresses based on the Autonomous System Number (ASN) to which they belong. It can be seen that many of the IP addresses belong to hosting companies in different parts of the world (e.g., SQUITTER, OVH, WINDTRE). We also see a wide-range sources from where the communication is established, some belonging to residential IP addresses.

Through our honeypot, we are able to characterize the different types of calls being made. We see that many of the call attempts are directed to phone numbers that may belong to real users, either launch spam or scam campaigns. Countermeasures such as IP address or phone number black-listing are not enough to prevent this type of abuse because cybercriminals can easily change the IP address (e.g., using TOR, VPNs, Proxy) or phone numbers used to carry out fraud campaigns (e.g., by using Caller-ID Spoofing or hacked Private Branch Exchange (PBX)) [3].

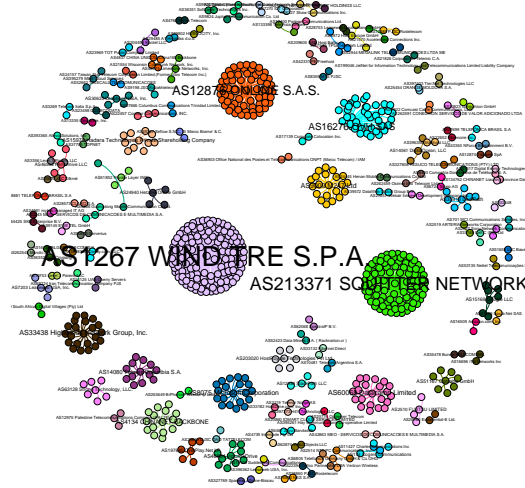


FIGURE 2. Grouping by ASN of all the IP addresses that interacted with the system.

In view of this, in the following section, we introduce a system geared towards automatically answering calls. The goal of system is twofold. First, we aim at learning whether the call is a scam by analyzing its content. This learning task can be placed in line with existing VoIP servers to tackle our first threat model (*Infrastructure*). We then aim at designing depleting the resources of the scammers by engaging them in a conversation with a chatbot for as long as possible. This reduces the time scammers can invest in the actual victims and addresses our second threat model (*Call Centers*).

### III. SYSTEM ARCHITECTURE

In this section we present the methodology followed in this research and the architecture of our system. We given an overview of our system, describe each component in detail, and we finally outline the ethical principles followed in this work.

#### A. OVERVIEW

Our system runs on a customized infrastructure built upon a Private Branch Exchange (PBX) based on Asterisk, which is a popular open-source communications framework. We embed custom processes into Asterisk to manage calls and inspect the conversation with the caller in real time. The pipeline works as follows: when a call is processed, we convert the incoming audio to text and then we generate an appropriate response in the same format. We then convert it to audio and play it back to the caller. At any given stage in during the phone call, we feed the conversation to a classifier to categorize the type of call received. Figure 3 shows the experimental architecture of our system, together with the workflow of a conversation and the different inputs

and outputs of each of the system components.

## B. OUR SYSTEM

In this section we describe the main components of our architecture.

**Speech to Text.** This component has the function of converting the audio of the speaker to text. It reads the audio channel of the conversation via the Asterisk Gateway Interface (AGI), and sends the raw audio to the speech recognition model. Different audio models (e.g., Vosk [13] or Google Speech-to-Text [14]) can be used. These work in a similar way, that is, by computing the audio in parallel and issuing a final response once the end of a sentence is detected. When the final sentence has been received in text format, it is sent to the chatbot component.

**Chatbot.** This module is the brain of our system and is responsible for maintaining the flow of the conversation with the fraudsters. Its architecture is made up of two types of chatbots. The first one is composed of a Natural Language Understanding (NLU) module built on top of Rasa NLU [15], which is an open-source framework for intent classification and entity extraction. It relies on Machine Learning (ML), but has a conversation-driven approach. Specifically, it allows to different intentions and entities and predefined responses with which the chatbot has been trained (i.e., greetings, call reason, listening issues, turning the computer on).

The second one is a chatbot built using OpenAI's Generative Pre-trained Transformer 3 (GPT-3) model [16]. This model was trained with 175 billion parameters and allows the performance of tasks with zero or very few examples. We use Rasa NLU to start the conversation and generate a context for the chatbot built on GPT-3. The start of telephone conversations is often very similar in different parts of the world, as if it were a rule or script that indicates how a call should begin [17]. For Rasa NLU's initial context training, we use intentions and entities that occur in any common telephone conversation (i.e., greetings or identification). Thus we use this model by giving it as context the start of the conversation produced by the model created in Rasa NLU, and use the generated response to feed back into the conversation. The answers generated are consistent with the conversation, allowing us to give valid replies regardless of the type of scam the fraudsters are trying to carry out. Therefore, this chatbot component receives the transcription of the audio generated by the caller as input, and that text is initially sent to one of the chatbots, either Rasa NLU or GPT-3. In the case of the Rasa NLU chatbot, it detects the type of call and we send back the response to the conversation in the form of pre-recorded audio that matches the intent of the response. On the other hand, when we use GPT-3, the responses generated by GPT-3 are open-ended and thus it is unfeasible to have it pre-recorded. We send this response to the text-to-speech component, convert it to audio, and play it back in the conversation, as described in the next.

**Text to Speech.** This module is responsible for converting the responses generated by the chatbot, from text to audio

TABLE 1. Number of audios distributed by category.

Class	# of Audios	Class	# of Audios
Tech support	58	Air duct	28
Electricity	57	Telecom services	25
Vacation	46	Loun	23
Alarm system	41	Windows&Doors	22
Home	40	Medical services	21
Fundraising	32	SEO	12
Credit card	29		

format, so that they can be played back in the conversation. The voice synthesis model works in a similar way than the transcriber (c.f. Speech to text). It obtains the text to be converted to audio as input, and a series of parameters such as the pitch, speed or gender of the voice of the audio that will be generated. This parameters are randomly chose to obtain a different voice in each of the conversations. To generate the audio in real time, we use the Text-to-Speech (TTS) library [18], which has different pre-trained models. In our case, we use the Tacotron2 [19] model together with the ParallelWaveGAN (PWGAN) vocoder [20] to perform the real-time speech synthesis.

**Scam and spam classifier.** This component is a machine learning model trained on a dataset of audio conversations with scammers we collect from a dedicated YouTube channel [21]. The dataset used for the model is composed of a set of conversations between spammers or scammers with an unsophisticated chatbot known as Lenny. Lenny plays the same set of audios in sequence one after the other one every time there is a silence in the conversation. First, we convert the audio to text using an offline speech recognition toolkit known as Vosk [13]. We use one of the pre-trained models of this toolkit for the English language, and a model for speaker recognition, which encodes the information of the speech expression in an embedding vector called x-vector [22], [23]. This makes it possible to calculate the cosine distance to determine whether the two vectors are similar and therefore come from the same speaker. Firstly, we obtain the x-vectors of all Lenny's audios through the model. Once all the vectors have been obtained, we use the model for transcribing the text by comparing the x-vector calculated for each generated text with those from Lenny's audio. If the cosine distance is close to 1, it means that the two vectors are similar and therefore correspond to the same speaker. In this way, we can accurately identify which parts of the audio are from the Lenny chatbot and which ones are from the scammer. In our case, we discard those from the Lenny chatbot. Once we have the transcripts of the utterances of the different scammers, we manually label the conversations according to the type of scam. In total, we have around 450 tagged audio conversations divided into 13 categories. Table 1 shows the categories and the number of audios that belong to each category. We divide the dataset into 80% for training and 20% for testing. This 20% of test data remains unseen and will be used to evaluate the model.

In a second stage, we pre-process the text by stemming the words of each of the audio transcripts in the training set.

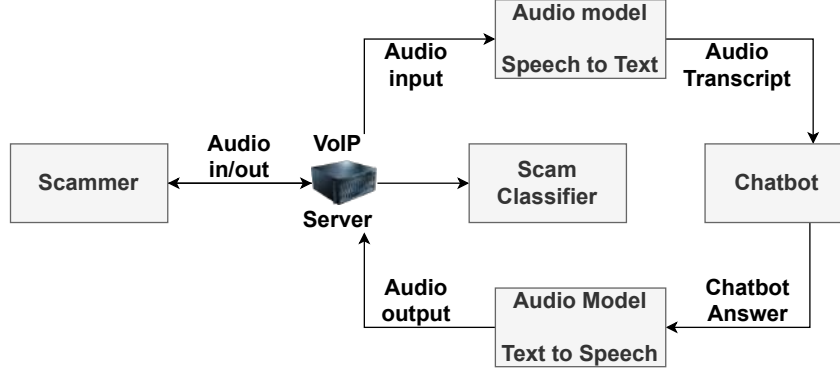


FIGURE 3. System architecture.

This stage also extracts the ngrams of size 1, 2 and 3 from each document to keep the context between various words in the documents. Then, we use Term Frequency-Inverse Document Frequency (TF-IDF) for feature extraction, which is used for measuring which words are most important in defining a document. TF measures the frequency of a term  $t$  in document  $d$ , as can be seen in Equation 1. IDF is used for calculating the weight of terms across all documents, that is, it indicates whether the term is common or not. Mathematically, IDF is the logarithm of the quotient between the total number of documents and the number of documents containing term  $t$ , as shown in Equation 2. This way, it is possible to give more importance to keywords for each category and less importance to generic words. Once TF and IDF are calculated, TF-IDF is computed as described in Equation 3.

$$tf_{t,d} = f_{t,d} \quad (1)$$

$$idf_{t,D} = \log \frac{|D|}{|\{d \in D : t \in d\}|} \quad (2)$$

$$tfidf_{t,d,D} = tf_{t,d} \cdot idf_{t,D} \quad (3)$$

The value obtained for each term will be between 0 and 1, where close to 0 indicates that the term is not relevant and appears in many documents, and close to 1 indicates that the term is important. Finally, we train a machine learning model to classify conversations using the extracted features and its previously calculated TF-IDF weight.

#### IV. ETHICS

This research uses [24] as a reference for the ethical principles that must be followed for Internet-mediated research, a work that extends the Belmont Report [25] and the Menlo Report [26]. Due to the type of research we carry in this work, we are unable to ask for informed consent to the scammers while carrying out the experiments. This is because the

output of our measurement would be biased if scammers were aware that they are talking with a chatbot. We address this challenge through a carefully dissing experiment and a risk-benefit assessment. First and foremost, we develop mechanisms to reduce the risk of the study and to avoid harming users. We then assess the benefits of our work and we note that our system would raise the bar in the mitigation of telephone scams through the Internet, reducing the scammers resources and with the intention that the time that they spend on a call with our chatbot is time that they do not invest on trying to scamming victims who may be vulnerable to this type of calls. Our experiments have been evaluated by our Institutional Review Board (IRB), who has considered the benefits and risks of the experiments and has approved our study. Below we detail the measures adopted for minimizing the risks of the study carried out.

First, we exhaustively tested our chatbot through a number of simulations to determine if outputs could be observed as offensive. This is particularly relevant for the chatbot that uses a generative transformer. We then limit the number of calls we connect through to our chatbot to the bare minimum necessary for evaluating the proposed system. Second, the phone numbers we use on our evaluation belong to the Toll-Free category, thus preventing scammers from obtaining any economic benefit from our calls. Thirdly, all the calls have been monitored in real time in order to verify that numbers are belong to scammers so that, if this was not the case, our experiment is interrupted as soon as possible. Finally, our experiments do not attempt to identify any of the scammers and we do not store any sensitive information about them. That is, the numbers we use to reach out scammers are extracted from open sources on the Internet. In addition, we do not record the conversations, we only store anonymously the transcripts of the audio obtained in real time.

**TABLE 2.** Results for the different types of machine learning algorithms.

Algorithm	Precision	Recall	FScore
K Neighbors	86.19	83.46	82.97
SVM kernel='rbf'	70.65	64.90	64.70
SVM kernel='Linear'	<b>94.46</b>	<b>90.39</b>	<b>91.04</b>
Decision tree	74.96	71.35	70.57
Random forest	91.72	85.29	86.16

## V. SPAM AND SCAM CLASSIFICATION

This section evaluates the effectiveness of the classification module presented in Section III.

### A. CLASSIFIER TRAINING

As mentioned in the previous section, we use as features the N-grams extracted from the conversations and their TF-IDF weight. We test different algorithms such as SVM [27], using the Linear and Radial Basis Function (RBF), Kernels [28], Decision Tree [29], Random Forest [30] and K-nearest [31]. We use the implementation of these algorithms provided by the Sklearn [32] library written in Python.

In the training phase, we use 80% of the initial dataset. We use K-fold [33] as a cross-validation strategy with the different machine learning algorithms discussed above. This training set has been divided into 3 stratified folds, that is, each of the folds has the same percentage of each class. For the evaluation of the algorithms, we use precision, recall and Fscore as metrics. We apply these metrics to each class separately, with the final result being the average of all classes. We choose this strategy so that each class contributes with equal weight to the results regardless of the total number of conversations they have. Table 2 shows the results obtained for each of the machine learning algorithms. It can be seen that the SVM algorithm with the linear kernel is the one that gives the best results in terms of precision, recall and FScore.

### B. CLASSIFIER EVALUATION (RQ1&2)

We train our classifier with the SVM linear kernel algorithm, using 80% of the data, and we validate it with the remaining 20%, whose data remains unseen. Figure 4 shows the confusion matrix for the SVM algorithm with the linear kernel. The confusion matrix details the correct and incorrect prediction percentage for each of the scam types and provides a picture of classification performance. We can see that for

most types of scam or spam, the model manages to classify the type correctly. However, for some typical types there are misclassifications (e.g., Loan as Fundraising, Alarm system as Medical services), although they do not exceed 20% for that class. In order to understand the reason for these misclassifications, we obtain the most correlated unigrams and bigrams for each type of scam or spam call. Table 3 shows the terms that have the highest correlation and, therefore, the greatest weight when classifying one type of call or another. It can be observed that the most relevant terms contain good information about the type of call they represent. However, in some cases they lead to misclassifications. For example, when observing the confusion matrix, we can see that some calls of the type Medical services are classified in the category of Credit cards. This is because the term card (one of the most related unigrams in Credit card) appears in some medical calls. Something similar occurs in the Alarm system type, where the caller says "I'm sure you're familiar with medical alarm systems", causing the classifier to classify that conversation as Medical services instead of Alarm systems. In the other cases, misclassifications occur due to reasons similar to those indicated above.

## VI. ONLINE EVALUATION

In this section, we evaluate our chatbot by means of real calls to scammers. Firstly, we describe the acquisition of the dataset, and then how we address the challenge of avoiding the interactive menus that some infrastructures have for avoiding automated calls. Finally, we present the results of the experiments.

### A. EXPERIMENTAL SETUP

To evaluate our system, we focus on the Technical support scam. We develop a specific chatbot for this type of scam, both for RASA and for the GPT-3 context, as discussed in Section III. There are different methods for obtaining successful calls, such as seeding phone numbers [34], so that these appear on telephone lists used by scammers, or obtaining dirty numbers (through some telephone company) that have been returned by customers because they received numerous unwanted calls [10]. Our system is designed to receive scam or spam calls once deployed in production. However, obtaining calls from scammers in a lab environment is challenging. Thus, we initially evaluate our system by making calls to numbers that are known by the community to belong to scammers. This presents an additional challenge as the turns of the conversation vary slightly with respect to those conversations in which the scammer calls the user. When the user receives the call from the scammer, the chatbot is in the ideal scenario: it is not necessary for the chatbot to take the initiative because it is the scammer who tries to convince the "user" to obtain a benefit. However, when the conversation is initiated by the "user", the chatbot is expected to provide the reason for the call. This is still a common scenario in practice, as some of these scammers use adware

**TABLE 3.** Most-correlated terms.

Type	Unigrams	Bigrams
Alarm system	secur, alarm	secur alarm, home secur
Air duct	air, duct	duct clean, air duct
Credit card	credit, card	lower rate, credit card
Energy	energ, electr	ga electr, direct energ
Fundraising	conserv, donat	parti canada, conserv parti
Home	project, paint	kitchen bathroom, home improv
Loan	busi, loan	cash advanc, busi loan
Medical services	knee, medicar	medic alert, insur polici
SEO	page, googl	busi owner, page googl
Tech support	technic, comput	turn comput, comput im
Telecom services	cabl, tv	phone internet, home phone
Vacation	resort, vacat	free cruis, vacat packag
Windows&Doors	door, replac	estim window, window door



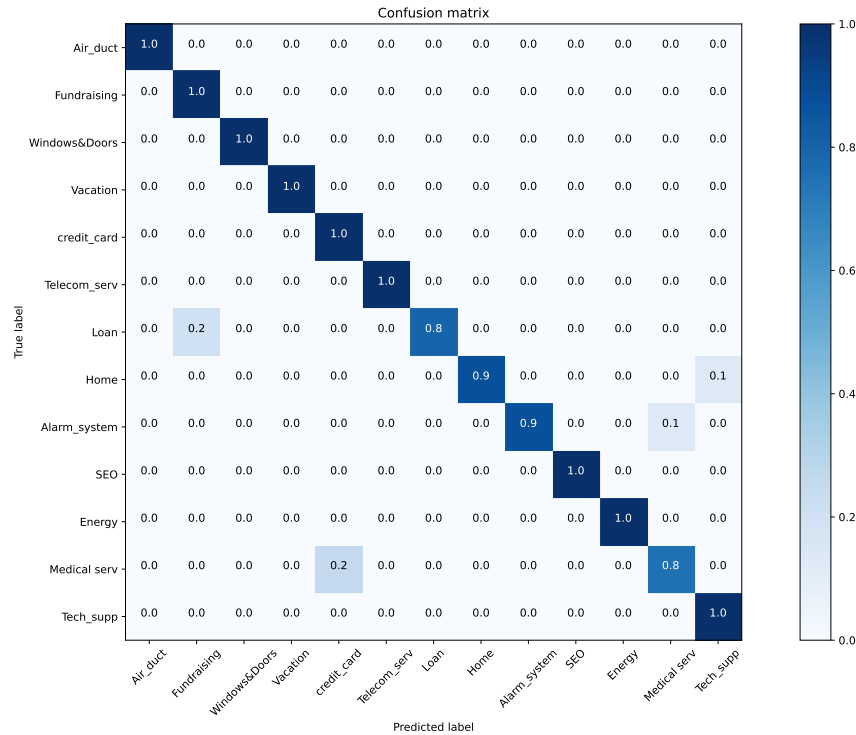


FIGURE 4. Confusion matrix for the SVM algorithm with linear kernel.

to advertise their phone numbers and wait for victims to dial in.

### B. PHONE NUMBER DATASET

Our dataset consists of a collection of telephone numbers reported as technical support scam collected from publicly available sources [35]–[37]. Since scammers change phone numbers and become unavailable quite regularly, we collect the numbers that were posted in 24-hour periods. We parse these phone numbers with the phonenumbers library [38], saving and calling those that belonged to the Toll Free numbers category, thus preventing the scammers from obtaining any economic return for the calls made. In total, we collected a total of 1,960 phone numbers for scammers.

### C. INTERACTIVE VOICE RESPONSE MENU

During the course of our experiments, we noticed that many calls were not reaching a live agent due to the use of an Interactive Voice Response (IVR) menu. These phone menus interact with users by offering a series of options before routing the call to a live agent. Therefore, in these cases the chatbot was not able to speak with a live agent. In order to increase the chances of success and reach a live agent, we

tried to bypass these interactive menus. Based on the calls we had made, we identified that most IVRs simply asked the caller to press a number on the phone to continue the conversation or be redirected to a technician. Therefore, we created a new intention on Rasa NLU, with the different messages that we had obtained from the interactive menus, so that it could be detected that we were dealing with an IVR. In this way, we can identify the numbers of the IVR menu and the possible options, and send a Dual Tone Multi-Frequency (DTMF) signal that simulates pressing a number on the phone. Fragment 1 shows an example of an IVR menu that is identified and where a DTMF signal is sent that allows the call to reach a live agent. In total, after including the bypass method, 46 menus were detected, and the number was correct in 38 of them, representing around 82.6%, allowing us to bypass simple IVR menus.

#### Fragment 1.

Support\_service: thank you for calling us press 1 for customer support press 2 for technical support  
Michael: [IVR\_MENU=2]

Support\_service: thank you for calling technical  
some hold for the next available agent  
Michael: hello How are you doing?

## D. RESULTS OVERVIEW (RQ3)

In order to evaluate our system, we make calls to the numbers that we collect every day and which belong to the Technical support type of scam. Table 4 shows a summary of the calls made and the status of the call with each of the chatbots. In total, 1,960 calls were made and 535 calls were answered, representing about 27% of the calls made. Of all the calls that

TABLE 4. Number of calls made.

Status of the call	Rasa chatbot	GPT-3 chatbot	Total
Busy	316	446	762
Answered	285	250	535
No Answer	321	264	585
Failed	57	21	78
Total	979	981	1,960

were answered, we discard those conversations that lasted less than two minutes since the vast majority of the calls that did not exceed that time did not succeed in being connected with a live agent, either because there was none available and they were kept on hold and hung up, or the system answered and hung up automatically. Figure 5 shows the number of calls considered valid and the duration of these calls distributed according to the type of chatbot used. The average duration using the chatbot created using Rasa NLU is around 4.2 minutes, while the average time for the chatbot using GPT-3 is around 3.8 minutes. In general, the chatbot

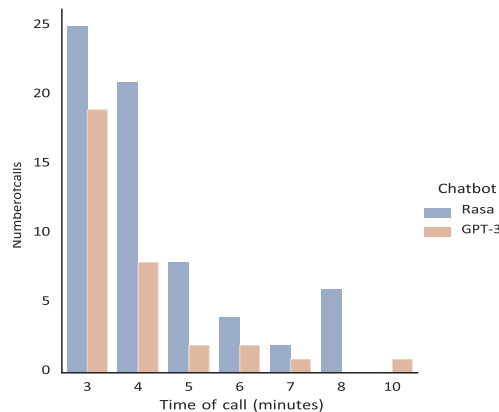


FIGURE 5. Duration of calls maintained by the chatbots.

built on Rasa NLU is able to hold conversations longer than the chatbot built on GPT-3. This is mainly because the Rasa chatbot is built with pre-recorded audio responses that are played back in the conversation, and, although it is not able to give coherent responses to statements it has not been trained

on, the audio is more reliable than that synthesized in real time in the GPT-3 chatbot, which sounds more robotic when the generated sentences are longer. In addition, the chatbot built with GPT-3 is more unpredictable within the context of the call, since it depends heavily on the transcript of the call. That is, if the transcript of the call is not accurate, the response can be very different from what is expected.

## E. ANALYSIS OF THE CONVERSATIONS (RQ4&5)

We next study the performance of both chatbots used. For this, we perform a qualitative study of the conversations they both generate in the look for scenarios where advantages or disadvantages appear in the use of the chatbot. In Fragment 2, the conversation maintained with the scammer through the Rasa NLU bot is shown. The main advantage of this bot lies in the use of categorical answers based on the analysis of the phrase received from the attacker, for example, those for greeting and explaining the initial problem, turning on the computer or explaining different errors that motivate the call. These categories address more than 15 different situations related to the context of the call, which leads the scammer to think that the call is not from a bot; firstly, because the answers are pre-recorded by a human (instead of being a synthetic audio from a bot), and, secondly, as the bot is capable of redirecting the conversation to its predefined categories when the transcription of the scammer's message is wrong. Thus, by resorting to phrases such as "Can you repeat the previous step?" or "I can't hear you well", the context of the call remains within the scope of the scam and does not lead to a totally different topic, thus maintaining the conversation with the scammer.

### Fragment 2. Rasa NLU

Michael: I have some errors on my computer. I think I have a virus and it is terribly slow.  
Support\_service: alright sir can you just tell me what exactly do you see on your screen right now  
Michael: what do i have to do now?  
Support\_service: yeah can you just tell me first of all let exactly what we were doing on your computer when this came up on your screen  
Michael: I have some warnings telling me my computer has viruses  
Support\_service: okay can you just tell me what exactly do you see on your screen right now  
Michael: Yes, thanks  
Support\_service: sorry I beg your pardon  
Michael: Are you there?  
Support\_service: yes I'm very sorry can you hear me  
Michael: Absolutely sir, I want to fix it  
Support\_service: okay just do one thing I will be connecting you with a secure server so just do one thing first of all just hit the Windows key on your

keyboard and the letter r with it alright

This does not happen in the case of GPT-3, since the answers are consistent with the transcription of the scammer's message, that is, they are not predefined. This is an advantage when answering with more fluid phrases, but a transcription error can mean a total change of context in the call, causing the fraudster to hang up. An example of GPT-3 is shown in Fragment 3, in which the responses given to the attacker are more realistic in terms of speech, but a transcription error of the word "water" leads to an unwanted context switch, which causes the scammer to end up terminating the conversation.

In summary, we can conclude that the Rasa NLU bot is more effective for the purpose of our paper. Designing a chatbot using Rasa NLU requires considerable larger efforts than GPT-3. In particular, it requires domain knowledge to build all possible conversational flows, requiring the study the different categories and their possible answers. It also requires to pre-record a range of answers for the category. Conversely, the results compared with GPT-3 are better with regards to the quality of the conversation and follow the flow of the conversation with the scammer for a longer time, depleting their resources and thus preventing the scammer from reaching other victims during that time interval. However, there are other reasons besides the quality of the response that can lead to ending the call with the scammer. These reasons are detailed in the next subsection.

### Fragment 3. GPT-3

Support\_service: you for calling to speak to a representative please press zero  
 John: [IVR\_MENU=0]  
 Support\_service: under for calling my name is Mark Courthouse  
 John: Yes, I see  
 Support\_service: hello  
 John: Hello, Mark Courthouse  
 Support\_service: tell me how may help you sir  
 John: I'm here because of last night's problem  
 Support\_service: what kind of problems  
 John: I accessed your homepage and my computer got infected with a virus  
 Support\_service: I understand sir tell me how may help you  
 John: By the way, I was calling from a phone-booth. When I use my computer I'm constantly stared at by strangers...  
 Support\_service: call really  
 John: Yes, I really like to  
 Support\_service: tell me how may help you  
 John: I'd like to be connected with the technician who is responsible for viruses.  
 Support\_service: we will be helping you in the

removing the wireless what kind of waters do you have

John: hello?

Support\_service: what kind of water do you have

John: I have hot

Support\_service: sorry

John: How may a hot

Support\_service: what did he said

John: I have water that's hot

#### 1) Main reasons for hanging up

Once the experiments were over, we analyzed the end of the calls to see the main reasons that lead scammers to hang up. These can be summarized as follows:

- The voice transcription is not quite correct, and the response generated is more unpredictable. Although it might actually make sense within the transcript, for the scammer it does not make sense, so they end up terminating the conversation. This occurs in the case of the GPT-3 bot, since in the case of the Rasa NLU bot the responses are pre-recorded, as discussed in the previous section.
- Our phone number does not appear on the lists for which they have launched the scam campaign. This reason is inherent in the type of chatbot used in the call since it only depends on the phone number used.
- Some responses generated seem technically more advanced and the audience that the scammers are trying to target is more vulnerable, that is to say, non-tech savvy users. Therefore, they understand that they will not obtain any benefit from the conversation and hang up. This occurs in the case of the GPT-3 bot, where sometimes the chatbot gives responses that are too "advanced" like "my hard drive keeps randomly freezing and frequently the blue screen of death pops up".
- They detect that it is a prank call to waste their time, mainly motivated by the audio generated in real time that sounds more robotic in long sentences. This is the case of the GPT-3 chatbot since the Rasa NLU chatbot uses audios that were pre-recorded by a human.
- Sentences from the scammer for which the bot has not been trained, giving rise to responses that it has not understood what the scammer has said, or that it did not hear the scammer well, etc, so when several occur in a row, it causes the scammer to end up terminating the call in most cases. This occurs for the Rasa NLU chatbot, where the model is trained to recognize a finite set of actions.

## VII. RELATED WORK

In this section we review the proposals from the community that address the same issue as this article, highlighting the state of the art in the different categories covered by our proposal.



**Honeypots.** Nawrocki *et al.* [39] carry out a systematic review of the pieces of research related to honeypot systems, describing the data that can be collected by them, and providing a classification based on the type of interaction they provide for attackers. Basically, there are three types of honeypot, offering low, medium and high degrees of interaction. The first of these presents a basic level of functionality of the service, while the last provides all the possible capabilities, making it more difficult for attackers to differentiate it from a real system. Medium interaction honeypots simulate certain capabilities of the service but do not offer full functionality. In the security community, honeypots are used to obtain information about the attacks carried out on these services in real time. There are numerous works focused on different protocols such as Secure SHell (SSH) [40], Telecommunication Network (Telnet) [41], [42] and Simple Mail Transfer Protocol (SMTP) [43] that collect data on attacks, malware, and Tactics, Techniques and Procedures (TTPs) that attackers use through these protocols.

With regards to the world of telephone communications, honeypots allow us to identify fraudulent behavior in telephone calls made either through conventional telephone lines or through VoIP communications. Gupta *et al.* [10] set up a low interaction honeypot system to obtain information on the types of attacks and abuses that are performed through the telephone line. In order to do so, they use a cloud-based infrastructure and around 39,000 telephone numbers (provided by a telephone provider) that had been returned by the provider's customers due to receiving unwanted calls. The system received a volume of 1.3 million calls from 250k unique phones in the 7 months that the honeypot was running, demonstrating the abuse that exists through mobile phone lines. A similar work is that of Marzuoli *et al.* [44], in which the authors collected around 40,000 unwanted calls, most of them made by just a few actors. They also found that most scam and spam calls could be identified on the basis of the audio features collected by the honeypot. Balduzzi *et al.* [45] developed the first honeypot intended for mobile devices. They seed the phone numbers to analyze attacks that occur on numbers that have not been used before. In total they received more than 2000 calls and text messages, most of them unwanted. Other works [46], [47] present honeypot architectures for the study and analysis of different attacks through the Session Initiation Protocol (SIP) used in VoIP communications. Both use the honeypot to detect specific attacks that occur in VoIP systems (i.e., DoS attacks, password brute force, and flooding).

**Scam and spam analysis.** Other studies focus on understanding the scam ecosystem by analyzing different types of scams. Miramirkhani *et al.* [48] analyze technical support scams through an automatic system that is capable of collecting information such as phone numbers and domains related to this type of scam. They make calls to these types of scammers to learn about the different tactics and procedures they use to deceive the victim. Mubarak *et al.* [49] carry out a review of the different activities of telephone scams that

have been carried out in Malaysia. They describe some of the scenarios that have occurred in Malaysia and the techniques used by scammers to achieve their goals. Finally, they propose two prevention mechanisms for protecting users against scams. Bidgoli *et al.* [50] analyze data from scam cases that have been reported on campus for three years. They provide qualitative information based on the type of scam, gender, the way it was reported, how they had been contacted, etc. Finally, they conducted surveys to understand the ecosystem of scams that affect international students and that led them to report these actions. Sahin *et al.* [8] analyze the effectiveness of a chatbot for combatting spam and scam calls. Their study is based on a bot known as Lenny, which is an Interactive Voice Response (IVR) system with a set of pre-recorded audios designed to waste the time of spammers in unwanted calls. In this work they analyze the calls received by Lenny as well as the transcribed conversations to understand how he is able to maintain interaction with spammers.

**Detection and classification.** Prasad *et al.* [7] categorize different types of audio and fraud campaigns by using a method of classifying the data and metadata of the calls. Unlike other pieces of research based on audio transcription, the method proposed in this work is language-independent, as it deals directly with the audios. Azad *et al.* [51] propose a method to classify users based on the interaction they have. To do this, they generate a social graph based on the Call Detail Record (CDR) and the relationship that the telephone numbers have. The edges have a weight based on the frequency of calls and the duration of these between the caller and the callee, representing the strength of that relationship. Finally, it generates a machine learning model with these characteristics extracted from the graphs for classifying whether a user is a legitimate user, spammer or a call center. Liu *et al.* [52] introduce a method to discover numbers involved in spam campaigns through detailed call logs. It does this by combining supervised and unsupervised learning methods to discover new numbers and add them to blacklists of phone numbers. Bordjiba *et al.* [53] develop a framework for detecting spam campaigns based on user complaints and the correlation between them in order to group spam campaigns and obtain information on this type of abuse. Li *et al.* [54] present a proposal based on machine learning to detect malicious calls. The proposed model uses a total of 29 features and is capable of blocking 90% of malicious calls. To generate the dataset, they developed a mobile app for users to tag malicious calls.

### VIII. CONCLUSION

In this proposal, an architecture capable of functioning as a honeypot against criminals who use phone calls to commit their offenses has been successfully deployed. Firstly, we built a spam/scam call classifier using different machine learning models. Among the different models for classifying calls, SVM with the linear kernel provided the best performance. Then, we built two chatbot models implement our study: on the one hand, Rasa with pre-recorded audio and, on

the other hand, GPT-3 with audio generated in real time. We also analyze the differences between them.

The results show that the chatbot built with Rasa NLU performs better in terms of engagement time with the scammers. As it is a rule-based chatbot, the answers are predefined and therefore can be pre-recorded. This inspires greater confidence in the scammer than an automatically-generated audio that can be recognized by the scammer as being a more robotic voice.

### ACKNOWLEDGMENT

This work has been supported by the MINECO and European Commission (FEDER funds) under project RTI2018-098156-B-C52, by the JCCM under the projects SBPLY/17/180501/000353 and SBPLY/21/180501/000195, by the Spanish Education, Culture and Sports Ministry under grant FPU 17/03105, and by the “Ramon y Cajal” Fellowship RYC-2020-029401-I.

### REFERENCES

- [1] M. Leonhardt, “Americans lost \$29.8 billion to phone scams alone over the past year,” Jun. 2021, section: Make It - Spend. [Online]. Available: <https://www.cnbc.com/2021/06/29/americans-lost-billions-of-dollars-to-phone-scams-over-the-past-year.html>
- [2] T. Williams, “New data shows phone scams are ‘exploding’ in Australia,” *ABC News*, Sep. 2021. [Online]. Available: <https://www.abc.net.au/news/2021-09-28/phone-scams-exploding-vulnerable-australians-new-data/100496496>
- [3] H. Tu, A. Doupé, Z. Zhao, and G.-J. Ahn, “Users really do answer telephone scams,” in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1327–1340. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity19/presentation/tu>
- [4] G. Suarez-Tangil, M. Edwards, C. Peersman, G. Stringhini, A. Rashid, and M. Whitty, “Automatically dismantling online dating fraud,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1128–1137, 2019.
- [5] H. Tu, A. Doupé, Z. Zhao, and G.-J. Ahn, “Sok: Everyone hates robocalls: A survey of techniques against telephone spam,” in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 320–338.
- [6] A. Kopylov, O. Seredin, A. Filin, and B. Tyshkevich, “Detection of interactive voice response (ivr) in phone call records,” *Int. J. Speech Technol.*, vol. 23, no. 4, p. 907–915, dec 2020. [Online]. Available: <https://doi.org/10.1007/s10772-020-09754-3>
- [7] S. Prasad, E. Bouma-Sims, A. K. Mylappan, and B. Reaves, “Who’s calling? characterizing robocalls through audio and metadata analysis,” in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 397–414. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity20/presentation/prasad>
- [8] M. Sahin, M. Relieu, and A. Francillon, “Using chatbots against voice spam: Analyzing lenny’s effectiveness,” in *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. Santa Clara, CA: USENIX Association, Jul. 2017, pp. 319–337. [Online]. Available: <https://www.usenix.org/conference/soups2017/technical-sessions/presentation/sahin>
- [9] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, “Advances in Knowledge Discovery and Data Mining,” in *From Data Mining to Knowledge Discovery: An Overview*. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996, pp. 1–34.
- [10] P. Gupta, B. Srinivasan, V. Balasubramanian, and M. Ahamad, “Phoneypt: Data-driven understanding of telephony threats,” in *NDSS. The Internet Society*, 2015. [Online]. Available: <http://dblp.uni-trier.de/db/conf/ndss/ndss2015.html#GuptaSBA15>
- [11] M. Gruber, C. Schanes, F. Fankhauser, and T. Grechenig, “Voice calls for free: How the black market establishes free phone calls—trapped and uncovered by a voip honeynet,” in *2013 Eleventh Annual Conference on Privacy, Security and Trust*. IEEE, 2013, pp. 205–212.
- [12] M. Sahin, A. Francillon, P. Gupta, and M. Ahamad, “Sok: Fraud in telephony networks,” in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2017, pp. 235–250.
- [13] AlphaCephei, “VOSK Offline Speech Recognition API,” 2022. [Online]. Available: <https://alphacephei.com/vosk/>
- [14] Google, “Speech-to-text: Automatic speech recognition,” 2022. [Online]. Available: <https://cloud.google.com/speech-to-text>
- [15] T. Bocklisch, J. Faulkner, N. Pawlowski, and A. Nichol, “Rasa: Open source language understanding and dialogue management,” 2017.
- [16] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.
- [17] E. A. Schegloff, “The routine as achievement,” *Human Studies*, vol. 9, no. 2/3, pp. 111–151, 1986.
- [18] Mozilla, “mozilla/TTS,” Apr. 2021. [Online]. Available: <https://github.com/mozilla/TTS>
- [19] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan, R. A. Saurous, Y. Ajiomvrgiannakis, and Y. Wu, “Natural tts synthesis by conditioning wavnet on mel spectrogram predictions,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 4779–4783.
- [20] R. Yamamoto, E. Song, and J. M. Kim, “Parallel wavegan: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram,” in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 6199–6203.
- [21] Taa.net, “Lenny!” [https://www.youtube.com/playlist?list=PLduL71\\_GKzHHk4hLga0nOGWrXlhl-i\\_3g](https://www.youtube.com/playlist?list=PLduL71_GKzHHk4hLga0nOGWrXlhl-i_3g), 2021.
- [22] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, “X-vectors: Robust dnn embeddings for speaker recognition,” in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2018, pp. 5329–5333.
- [23] D. Snyder, D. Garcia-Romero, A. McCree, G. Sell, D. Povey, and S. Khudanpur, “Spoken language recognition using x-vectors,” in *Odyssey*, 2018, pp. 105–111.
- [24] I. F. Anabo, I. Elexpuru-Albizuri, and L. Villardón-Gallego, “Revisiting the Belmont Report’s ethical principles in internet-mediated research: perspectives from disciplinary associations in the social sciences,” *Ethics and Information Technology*, vol. 21, no. 2, pp. 137–149, Jun. 2019. [Online]. Available: <https://doi.org/10.1007/s10676-018-9495-z>
- [25] V. A. Miracle, “The belmont report: The triple crown of research ethics,” *Dimensions of Critical Care Nursing*, vol. 35, no. 4, pp. 223–228, 2016.
- [26] D. Dittrich, E. Kenneally et al., “The menlo report: Ethical principles guiding information and communication technology research,” US Department of Homeland Security, Tech. Rep., 2012.
- [27] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, “Support vector machines,” *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, July 1998.
- [28] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*. Burlington, MA: Morgan Kaufmann Publishers Inc., 2011.
- [29] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Elsevier, Jun. 2014, google-Books-ID: b3ujBQAAQBAJ.
- [30] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [31] J. Laaksonen and E. Oja, “Classification with learning k-nearest neighbors,” in *Proceedings of International Conference on Neural Networks (ICNN’96)*, vol. 3, June 1996, pp. 1480–1483 vol.3.
- [32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [33] M. Stone, “Cross-Validatory Choice and Assessment of Statistical Predictions,” *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 36, no. 2, pp. 111–147, 1974. [Online]. Available: <https://www.jstor.org/stable/2984809>

- [34] M. Balduzzi, P. Gupta, L. Gu, D. Gao, and M. Ahamad, "Mobipot: Understanding mobile telephony threats with honeycards," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016, pp. 723–734.
- [35] thunder, "Latest tech-support-scam topics," 2021. [Online]. Available: <https://scammer.info/tag/tech-support-scam>
- [36] Bobrtc, "The Telephone Fraud Scam Database," 2021. [Online]. Available: <https://bobrtc.tel>
- [37] choozn, "Research into Tech-Support-Scams | PopUpDB.org," 2021. [Online]. Available: <https://popupdb.org>
- [38] D. Drysdale, "phonenumber: Python version of Google's common library for parsing, formatting, storing and validating international phone numbers," 2021. [Online]. Available: <https://github.com/daviddrysdale/python-phonenumbers>
- [39] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, "A survey on honeypot software and data analysis," *CoRR*, vol. abs/1608.06249, 2016. [Online]. Available: <http://arxiv.org/abs/1608.06249>
- [40] D. Fraunholz, D. Krohmer, S. D. Anton, and H. D. Schotten, "Investigation of cyber crime conducted by abusing weak or default passwords with a medium interaction honeypot," in *2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security)*. IEEE, 2017, pp. 1–7.
- [41] H. Šemić and S. Mrdović, "Iot honeypot: A multi-component solution for handling manual and mirai-based attacks," in *2017 25th Telecommunication Forum (TELFOR)*, 2017, pp. 1–4.
- [42] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: Analysing the rise of iot compromises," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.
- [43] L. Zobal, D. Kolár, and J. Kroustek, "Exploring current e-mail cyber threats using authenticated smtp honeypot," in *ICETE (2)*, 2020, pp. 253–262.
- [44] A. Marzuoli, H. A. Kingravi, D. Dewey, and R. Pienta, "Uncovering the landscape of fraud and spam in the telephony channel," in *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2016, pp. 853–858.
- [45] M. Balduzzi, P. Gupta, L. Gu, D. Gao, and M. Ahamad, "Mobipot: Understanding mobile telephony threats with honeycards," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 723–734. [Online]. Available: <https://doi.org/10.1145/2897845.2897890>
- [46] R. do Carmo, M. Nassar, and O. Fester, "Artemisa: An open-source honeypot back-end to support security in voip domains," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, 2011, pp. 361–368.
- [47] M. Nassar, S. Niccolini, R. State, and T. Ewald, "Holistic voip intrusion detection and prevention system," in *Proceedings of the 1st International Conference on Principles, Systems and Applications of IP Telecommunications*, ser. IPTComm '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 1–9. [Online]. Available: <https://doi.org/10.1145/1326304.1326306>
- [48] N. Miramirkhani, O. Starov, and N. Nikiforakis, "Dial one for scam: Analyzing and detecting technical support scams," *CoRR*, vol. abs/1607.06891, 2016. [Online]. Available: <http://arxiv.org/abs/1607.06891>
- [49] M. F. Mubarak, S. Yahya, and A. F. A. Shaazi, "A review of phone scam activities in malaysia," in *2019 IEEE 9th International Conference on System Engineering and Technology (ICSET)*, 2019, pp. 441–446.
- [50] M. Bidgoli and J. Grossklags, "'hello, this is the irs calling.': A case study on scams, extortion, impersonation, and phone spoofing," in *2017 APWG Symposium on Electronic Crime Research (eCrime)*, 2017, pp. 57–69.
- [51] M. A. Azad, M. Alazab, F. Riaz, J. Arshad, and T. Abullah, "Socioscope: I know who you are, a robo, human caller or service number," *Future Generation Computer Systems*, vol. 105, pp. 297–307, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X19315110>
- [52] J. Liu, B. Rahbarinia, R. Perdisci, H. Du, and L. Su, "Augmenting telephone spam blacklists by mining large cdr datasets," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ser. ASIACCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 273–284. [Online]. Available: <https://doi.org/10.1145/3196494.3196553>
- [53] H. E. Bordjiba, E. B. Karbab, and M. Debbabi, "Data-driven approach for automatic telephony threat analysis and campaign detection," *Digital Investigation*, vol. 24, pp. S131–S141, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1742287618300483>
- [54] H. Li, X. Xu, C. Liu, T. Ren, K. Wu, X. Cao, W. Zhang, Y. Yu, and D. Song, "A machine learning approach to prevent malicious calls over telephony networks," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 53–69.

\*\*\*



## CHAPTER 5

# Automatic Analysis Architecture of IoT Malware Samples

- **Title:** Automatic Analysis Architecture of IoT Malware Samples.
- **Authors:** Javier Carrillo-Mondéjar, Juan Manuel Castelo Gómez, Carlos Nuñez-Gómez, Jose Roldán Gómez and José Luis Martínez.
- **Type:** Journal paper.
- **Journal:** Security and Communication Networks
- **Publisher:** Hindawi.
- **ISSN:** 1939-0114.
- **Status:** Published.
- **Publication date:** October 2020.
- **Volume:** 2020.
- **Paper Number:** 8810708.
- **DOI:** 10.1155/2020/8810708
- **JCR IF/ranking:** 1.791/Q4 (JCR2020).

## Research Article

# Automatic Analysis Architecture of IoT Malware Samples

**Javier Carrillo-Mondejar , Juan Manuel Castelo Gomez, Carlos Núñez-Gómez, Jose Roldán Gómez, and José Luis Martínez**

*Research Institute of Informatics (I3A), Universidad de Castilla-La Mancha, Albacete 02071, Spain*

Correspondence should be addressed to Javier Carrillo-Mondejar; [javier.carrillo@uclm.es](mailto:javier.carrillo@uclm.es)

Received 30 March 2020; Revised 16 July 2020; Accepted 7 October 2020; Published 26 October 2020

Academic Editor: Yin Zhang

Copyright © 2020 Javier Carrillo-Mondejar et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The weakness of the security measures implemented on IoT devices, added to the sensitivity of the data that they handle, has created an attractive environment for cybercriminals to carry out attacks. To do so, they develop malware to compromise devices and control them. The study of malware samples is a crucial task in order to gain information on how to protect these devices, but it is impossible to manually do this due to the immense number of existing samples. Moreover, in the IoT, coexist multiple hardware architectures, such as ARM, PowerPC, MIPS, Intel 8086, or x64-86, which enlarges even more the quantity of malicious software. In this article, a modular solution to automatically analyze IoT malware samples from these architectures is proposed. In addition, the proposal is subjected to evaluation, analyzing a testbed of 1500 malware samples, proving that it is an effective approach to rapidly examining malicious software compiled for any architecture.

## 1. Introduction

The appearance of the Internet of Things (IoT) has greatly improved the application of technology in the everyday lives of people. Years ago, digital interaction between an individual and technology was in general only through a computer. With the development of smartphones, that communication became a more mobile, personal, and continuous task. And then, the IoT appeared to change all the previous concepts and insert technology into almost every imaginable object. Smart houses, eHealth, or smart cities are just a few examples of contexts that have their origin in the application of the IoT. Thus, not only has it helped to complement existing scenarios but it has also given rise to the ones in which technology is applied.

As a consequence, the volume of data that is now digitally handled has vastly increased as well. However, although the emergence of the IoT has clearly benefited people, the same positive verdict cannot be passed when speaking of the security measures implemented on the devices. Unfortunately, developers opted to prioritize

usability over security, especially during the IoT's conception, when the thought of someone compromising an entire network by simply attacking a switch was unthinkable.

Therefore, there was a huge underestimation of the requirements that these devices and the information that they handle demand. Nowadays, this issue is being acknowledged, and companies are working on improving the protection, but they are still quite vulnerable, added to the fact that a great number of old devices is still being used. This makes the IoT the perfect environment for cybercriminals to operate in. They can gain access to very sensitive and valuable information with little effort. Recent studies [1] show the magnitude of the problem. Only in the first quarter of 2019, a hundred million attacks were detected on smart devices, a figure seven times greater than the number found in 2018. Unsurprisingly, the *Mirai* malware family was behind 39% of them, taking advantage of old devices with unpatched vulnerabilities. Another sample which exploits a trivial attack, namely, the brute-force, *Nyadrop*, closely followed *Mirai* and reached a percentage of 38.57%.

These attacks were the result of poorly designed security measures on the devices and could have been easily mitigated by just changing the default user and password of the device for a more secure one. Instead, they ended up affecting companies such as Twitter, Amazon, Spotify, and Netflix, costing them millions of dollars and affecting their customer's trust [2].

As mentioned above, most IoT attacks do not have their origin in new malware samples, but are based on previous ones that were successful. New versions of old attacks appear every day with minor modifications, but the way they work remains almost identical. Having information about how a sample interacts with the compromised device, and what actions it carries out, allows investigators to protect the device or, at least, limit its expansion over the network. For this reason, the ability to identify which malware samples are alike, that is, those that belong to the same family, can have a huge impact when determining what actions to be taken in order to reduce the impact of a cyberincident.

In addition, besides the existence of multiple operating systems, there are also several architectures used by IoT devices, such as ARM, PowerPC, MIPS, and x86. With the aim of expanding the range over which cybercriminals can carry out their attacks, they develop samples for more than one. This means that numerous pieces of malware have their origin in a sample, and then it is adapted to work on other architectures. Consequently, its behaviour remains similar, with only its structure varying in order to be compatible with them. This allows the malware analyst to analyze malware families independently of the architecture for which the sample was designed.

This analysis is neither a trivial task nor a speedy one. The number of existing samples, added to the appearance of new ones almost every minute, makes it impossible for an investigator to study all of them. Therefore, it is necessary to develop automatic solutions, such as architectures or frameworks, which can speed up the process and be able to examine multiple samples at once. In order to achieve that, a change of approach is needed: instead of focusing on the features that differentiate a sample, now it is mandatory to determine which characteristics allow a piece of malware to be grouped with another, as well as selecting the ones that can be collected and interpreted automatically.

Therefore, the contributions of this study are as follows:

We study the current state of malware analysis, focusing on the development of automatic solutions to perform examinations

We present a series of static and dynamic characteristics that are useful to automatically categorize malware samples

We propose a modular framework for the automatic analysis and clustering of malware samples from the most widely used architectures, based on the evaluation of their static and dynamic features

We evaluate the proposal with a testbed of nearly 1,500 pieces of malware, confirming its usefulness when analyzing and clustering samples from different IoT architectures

The rest of the paper is organized as follows. Section 2 describes the IoT's architecture, its malware threats, and how to obtain useful characteristics from them. An architecture to automatically cluster malware samples from different IoT architectures is presented in Section 3. An evaluation of the proposal through the analysis of 1500 malware samples is carried out in Section 4. Finally, our conclusions are presented in Section 5.

## 2. Background

As discussed in the previous section, the IoT environment is the perfect target for cybercriminals to attack. This section presents the problem related to the large number of devices with different architectures connected to the Internet, lists the reasons for the rise of IoT security threats, and defines the concepts of malware analysis and characterization. Then, the Service-Oriented Architecture (SOA) software paradigm used in the design of the framework is introduced. In addition, we present a review of the proposals from the research community in regard to this paper.

**2.1. The IoT Environment.** The IoT allows developers to model use cases that in the past were not feasible due to the specific limitations of traditional client-server architectures: resource centralization, expensive devices, and high latencies, among others. The IoT environment creates room for new contexts such as Industry 4.0 [3] and smart homes [4]. Its structure can be divided into three fundamental building blocks: the Cloud Layer, the Network Layer, and the Devices Layer. Figure 1 shows the hierarchy formed by these layers. Frequently, end devices interact with other IoT devices as well as with large data centers in the cloud layer to carry out the tasks (sometimes computationally intensive ones) assigned to these end devices. Accordingly, more and more end devices are exposed to the Internet every day, so it is important to adopt appropriate security measures if we do not want to expose our end devices to external attackers.

Another main problem of the IoT environment is the considerable heterogeneity of the devices that comprise it. Although it is important to define security, analysis, and clustering mechanisms against malware layer by layer, our work focuses on the constrained-resource devices of the device layer. These devices are built with different hardware specifications and run different operating systems. One of the most significant specifications is the processor architecture used by such devices. Each processor and its instruction set are designed in a specific way. For example, ARM is a more energy-usage-concerned architecture than x86-64. In our case, the proposed framework focuses specifically on modelling Intel 80386, x86-64, MIPS, ARM, and PowerPC architectures.

**2.2. Threats.** By scrutinizing the aforementioned recent studies focused on evaluating new trends in IoT malware, a drop in the number of attacks via Telnet can be observed for the second quarter of 2019. Now, the value almost reaches 60%, 20% less than in the previous one. This



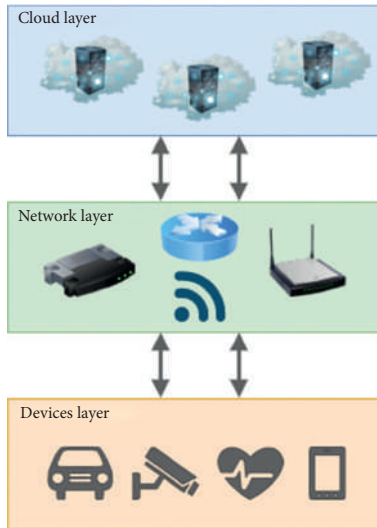


FIGURE 1: IoT environment architecture.

statistic can be seen as an encouraging one if we deduce that the decrease was due to developers no longer using that service, which is well-known to be deprecated and unsafe. The most worrisome data are that there are few changes in the most common malware families with respect to previous years, meaning that old attacks are still being successful. In addition, the number of malware samples is still growing and expanding into more areas [1]. Some of the main causes of the rapid growth in cybercrime in the IoT are the following:

**Number of connected devices:** during the year 2020, this figure is forecasted to reach 20.4 billion [5], with 5.8 billion of them being used in the enterprise and automotive market [6]. This means that there are more IoT devices than conventional ones, e.g., smartphones or computers. Therefore, it is preferable for cybercriminals to perform large-scale attacks in this environment rather than in the traditional one, as they can target more victims.

**Implemented security measures:** as briefly mentioned above, IoT devices can be easily compromised by carrying out simple brute-force or dictionary attacks. This is mainly due to the usage of weak default login credentials. Although it may seem ludicrous, the combination of user and password such as “admin-admin” or “admin-1234” is not that uncommon.

**Data handled:** the application of the IoT has led to the generation of data that previously did not exist or only did so in a smaller quantity. eHealth is a good example of this circumstance: metrics such as heart rate, blood pressure, or oxygen levels were only stored in special facilities such as hospitals or medical centers and were only available to restricted personnel. Nowadays, these

data are also measured and stored by smart watches or smart bracelets that are connected to the cloud and create personal profiles for each user.

Limited computational capacity of the devices: this makes them easy to crash, which is quite convenient when a cybercriminal wants to perform a DoS (Denial of Service) attack. The number of petitions that can be handled by these devices is far more limited than in conventional ones. In addition, it hinders the task of using antiviruses or cryptography algorithms, since the current versions are only supported by more powerful devices.

**2.3. Malware Characterization.** Characterization can be explained as a process in which a set of features are extracted from someone or something. This makes it possible to describe each item in an unambiguous way. Thus, malware characterization is the process of identifying and extracting these features from each malicious sample. In this field, the characteristics are divided into the following categories:

**Static features:** here, the focus is on the analysis of the intrinsic characteristics of a binary file without executing its code in the system. Information such as the strings that appear in it, its sections, architecture, opcodes, cyclomatic complexity, or entropy belongs to this category. The main advantage is that static characteristics are quick to extract automatically. On the other hand, the usefulness of the features may be affected if the sample is packed or obfuscated (i.e., disassembly code and strings).

**Dynamic features:** here, the target is the analysis of the behavior of the sample at runtime by monitoring the different actions that it carries out in the system. The data are extracted from the communication that the malware performs through the network and its interaction with the system, such as system calls or open files, among others. One of its disadvantages is that only characteristics of the executed portions of code are captured, so the criminals include monitoring detection techniques that prevent the sample from executing entirely. In addition, the extraction of dynamic features is more time consuming than the retrieval of static features due to the fact that the sample must be executed for a short period of time.

**2.4. SOA.** SOA is a software design paradigm in which modules work as independent services providing a specific interface to be called upon. They communicate through an Enterprise Service Bus (ESB) which is formed of one or several protocols, allowing the addition of services with little effort. In order to call each service when it is needed, an orchestration process is used [7]. Under this scheme, it is possible to add new components or new protocols. In addition, this architecture allows the easy integration of multiple SOA-based applications.



**2.5. Related Work.** As far as the authors are aware, there are no approaches available in the literature that jointly tackle the task of analyzing large numbers of malware samples specifically designed for the IoT and that of classifying or clustering them. On the contrary, most of the approaches try to describe specific malware samples or families, as mentioned in Section 2.5.1. In terms of automatically analyzing a great number of malware samples, there are some articles, but they focus only on Linux-based operating systems for x86 architectures, as is shown in Section 2.5.2. Finally, Section 2.5.3 covers approaches focused on classifying IoT malware, but these do not take into account all IoT architectures or families and neither do they study both static and dynamic features.

**2.5.1. Malware Survey.** Pa et al [8] presented a Telnet honeypot for different IoT architectures. They conducted a study of the malware that was aimed at this service, showing the problem that it suffers from when it is accessible from the Internet. The authors also presented the first sandbox that supported different architectures and executed the binaries and commands received through their honeypot.

Cozzi et al. [9] presented a complete malware study aimed at Linux-based operating systems. They statically and dynamically analyzed more than 10,000 samples distributed among the main architectures, namely, ARM, PowerPC, and MIPS, among others. They presented the main techniques used by malware and numerically expressed their use in the samples that made up their dataset. To carry out their analysis, they introduced the first malware analysis framework aimed at analyzing Linux-based malware.

Costin et al. [10] introduced a study of 60 families of IoT malware. The authors studied the timeline of events related to each family as well as the most relevant vulnerabilities used by them. For the dynamic analysis, the authors presented a sandbox compatible with the main IoT architectures based on the open source project Cuckoo Box [11].

**2.5.2. Linux-Based Sandbox.** Limon [12] is a sandbox for analyzing Linux-based malware. It collects calls to the operating system as well as capturing network traffic. Its main problem is that it only supports binary analysis in x86 architectures, and the operating system used to perform dynamic analysis is based on Ubuntu, which is not a very common operating system in the IoT. Similar problems are present in Detux [13], which, although it supports five architectures, is based on the Debian operating system. Detux only performs basic static analysis and network analysis, ignoring malware behavior within the operating system.

Chang et al. [14] proposed a sandbox for analyzing malware samples in the IoT. It is able to collect network packages and malware behavior in the system. To test the functionality of their sandbox, they experimented with the Zollard botnet.

**2.5.3. Classification.** Nghi Phu et al. [15] presented a framework for analyzing and classifying malware in the IoT. Their framework supports the MIPS architecture and

extracts features related to malware interaction with the system in order to train a machine learning model.

Alhanahnah et al. [16] suggested a new approach to classifying IoT malware compiled for different architectures. Its method is based on generating signatures at a high level since these are more robust and vary less between architectures.

Su et al. [17] introduced a method for malware classification in IoT environments. It is based on converting malware into an image and a convolutional neural network for classification. It is able to classify a sample into malware or goodware and recognizes two malware families: Mirai and Gafgyt.

Kumar et al. [18] proposed a new approach to differentiate between malicious and benign applications based on a ranking of permissions used in Android IoT devices. Their methodology included an improvement on the random forest algorithm, achieving an increase in the accuracy of malware detection.

Lei et al. [19] presented a system for malware detection on Android-based IoT devices. They proposed the use of event groups instead of API calls to capture malware behaviour at a higher level than in API level. They trained and evaluated their system with a dataset of around 15,000 and 29,000 benign and malicious Android apps, respectively.

## 3. Proposed Architecture

This section describes the proposed SOA-based modular framework for analyzing and classifying malware samples from different IoT architectures. It consists of six modules which are invoked as services by the orchestrator of the system, which is responsible for using each module and processing the information extracted in each of the stages. Due to its modular structure, each of the modules that make up the system can be used independently (i.e., deploying a virtual machine to execute commands from a honeypot or even for adding new components). These services use our Enterprise Service Bus (ESB), which allows us to integrate any new component easily. Figure 2 shows a global view of our architecture.

**3.1. System Overview.** The system uses an executable file from any of the architectures supported as input, analyzes it, and produces a cluster based on the similarity that it has with other previously examined files as output. Although the proposal is designed for malware analysis purposes, it is valid for clustering other types of executables. The following sections describe in detail the modules of which our system is composed.

**3.2. The Orchestrator.** This is the main module of the system and the one in charge of making the pipeline that interconnects the rest of the modules. Once it obtains a sample, it uses the static analysis module to obtain the information necessary to continue with the next phase. Then, it uses the deployment module to check whether the architecture of the analyzed file is supported, that is, whether there is a virtual

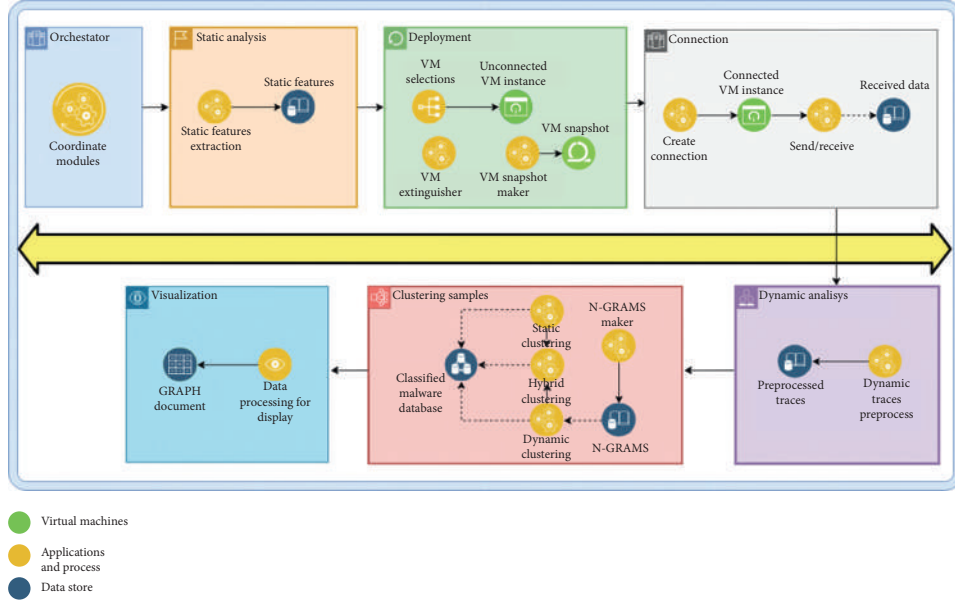


FIGURE 2: The proposed architecture for the analysis and clustering of IoT malware.

machine that supports that architecture, and if it is, it starts the virtual machine instance.

Once the virtual machine is on, it connects to it through the connectivity module and then proceeds with dynamic analysis, executing the file with the monitoring tool indicated in the configuration files. Then, the file is executed for a certain time which is indicated through the configuration commands of the framework. Once that timeout has elapsed, it obtains the result in the form of execution traces, destroys the virtual machine, and recovers the previous snapshot of the machine.

Finally, it calculates the similarity with other analyzed samples and adds it to the corresponding cluster if the similarity index is greater than the established threshold. The sample will be added to the cluster in which the most similar sample is located. If the threshold is not reached, a new cluster will be created to include the analyzed file.

Additionally, if the display parameter is active, it will calculate the similarity between all the samples and generate a graph connecting all of them.

**3.3. Static Analysis.** This module is responsible for obtaining and parsing the Executable Linkable Format (ELF) files. It is built upon *radare2* [20], a reverse engineering suite, and automates the process of obtaining information contained in the headers of the ELF files, as well as data regarding their sections. The static analysis module collects the following information.

**Information file:** characteristics of the headers of the executable file, such as architecture, whether the binary has

been stripped of the symbols or not, and whether it was compiled with static or dynamic libraries.

**Entropy:** this measures the lack of predictability of a data set. In binary analysis, a high entropy value indicates that the sample is obfuscated or packed.

**Cyclomatic complexity:** this is a metric used in software engineering to calculate, in a quantitative way, the complexity at a logical level of a program or function [21]. Cyclomatic complexity is calculated for each of the functions found in the disassembled code.

**Opcodes:** the sequence of operation codes (opcodes) of all the functions present in the disassembly of the program are extracted and stored.

**Libraries:** the name of the shared libraries used by the program.

**Sections:** the sections into which the executable is divided are extracted, also determining their permissions and entropy.

**Functions:** the name of the functions imported from the libraries and used by the program.

**Strings:** all text strings present in the sample.

**Hash:** the hash to uniquely identify the executable.

**3.4. Deployment Module.** This module is responsible for starting the virtual machine, shutting it down, or restarting it. Its input is the architecture for which the malware was developed, which is searched for in the library in order to determine whether it can be emulated or not. It uses *libvirt*

[22] to manage the virtualization platforms and the QEMU [23] emulator as hardware virtualizer. To emulate an architecture, it has to be supported by QEMU, and a guest domain in an eXtensible Markup Language (XML) must be defined. This file contains the configuration of the machine in libvirt, that is, its storage, CPU architecture, kernel image, and network properties. Once the machine has been started, the module returns a handler, which allows you to shut down or restart the machine as well as to see which machines are currently active. Finally, when a machine is stopped, a previous snapshot of the machine is recovered in order to have a malware-free image for the next analysis. In this way, this module provides the flexibility to add user-defined virtual machines and uses them in our framework.

**3.5. Connectivity and Dynamic Analysis.** This is the module responsible for establishing connection with the virtual machine. It allows the upload and download of files through the Secure Copy Protocol (SCP) and the execution of commands through the Secure SHell (SSH). It provides the flexibility to upload any file type and execute commands in the virtual machine. For example, it can upload an executable file or script and use any type of monitoring tool available in the virtual machine for extracting information about its behavior, such as *strace* [24] or *systemtap* [25]. Finally, download the monitored traces and parses the collected data. The parsing function is responsible for extracting the executed syscalls from the execution traces as well as their parameters and results. Table 1 shows an example of a run sequence and the syscall data.

**3.6. Clustering of Samples.** This is in charge of clustering the binary files based on some of the previously extracted features. Given two executable files, it calculates the index of similarity between them and, if this is greater than a set threshold (set through the configuration parameters), these samples are considered to be related and, therefore, will be part of the same cluster. To calculate the similarity, the module uses the following approaches:

*Dynamic approach.* We use the execution traces obtained in the dynamic analysis to generate sequences of *syscall* names of size  $n$  (set through the configuration parameters), which are known as  $n$ -grams. An example for a sequence of size  $n = 4$  is shown in Table 1, resulting in the following set of  $n$ -grams: (*brk*, *socket*, *fcntl64*, and *fcntl64*), (*socket*, *fcntl64*, *fcntl64*, and *setsockopt*), and (*fcntl64*, *fcntl64*, *setsockopt*, and *brk*). In order to determine the similarity, we use the Jaccard index [26] as a metric, which, for two sets of  $n$ -grams, is calculated as

$$\text{jaccard}(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|}, \quad (1)$$

TABLE 1: Format execution trace.

Syscalls	Parameters	Results
Brk	0x32000	0x32000
Socket	AF_INET, SOCK_RAW, IPPROTO_TCP	0
fcntl64	0, F_GETFL	0x2
fcntl64	0, F_SETFL, O_RDWR O_NONBLOCK	0
Setsockopt	0, SOL_IP, IP_HDRINCL, [1], 4	0
Brk	0x33000	0x33000

where the numerator indicates the number of unique subsets that are present in both sets, and the denominator indicates the total number of unique subsets between  $s_1$  and  $s_2$ . The result is a value between 0 and 1 which indicates the degree of similarity between two sets of  $n$ -grams.

*Static approach.* We use two metrics to measure the similarity between two executable files. The first is based on sequences of opcodes of size  $n$  extracted from the disassembled code. This is calculated in the same way as in the dynamic approach but using opcodes instead of *syscalls*. The second is based on the cyclomatic complexity of each of the functions present in the disassembled binary. A distance function is used for the calculation of the similarity between two executable files. This function is formalized as follows:

$$\text{distance}(s_1, s_2) = \sum_{i=0}^{|F|} \frac{\min(f_i^{s_1}, f_i^{s_2})}{\max(f_i^{s_1}, f_i^{s_2})} \times \frac{1}{F} \quad (2)$$

For example, let us consider two executables with five and seven functions, the first with cyclomatic complexities 3, 5, 3, 7, and 4 and the second with complexities 3, 3, 6, 4, 5, and 2. The first sample has two functions with cyclomatic complexity 3, one with 5, one with 7, and another with 4. In the second sample, we have two functions with cyclomatic complexity 3, two with 6, one with 4, one with 5, and another with 2. We normalize the vectors so that they have the same number of elements, and the vectors (0, 2, 1, 1, 0, 1) and (1, 2, 1, 1, 2, 0) are obtained. Therefore, the similarity index between the two vectors is 0.5 and is calculated as follows:  $((0/1 + 2/2 + 1/1 + 1/1 + 0/2 + 0/1)/6)$ .

*Hybrid approach.* The hybrid approach allows clustering using the indexes described above. To do this, it assigns a weight to each of the indexes to calculate the final similarity index. The weight of each index can be configured in the framework configuration files.

**3.7. Visualization.** Its function is to visually represent the groupings generated based on the approaches described above. We denote  $f$  as a function that defines whether two malware samples are similar or not using the following expression:

$$f(x = \text{metric}(s_1, s_2) | s_1, s_2) = \begin{cases} 1, & x \geq z \\ 0, & x < z \end{cases}; z \quad x \in [0, 1]; s_1, s_2 \in D, \quad (3)$$

where  $z$  being the selected threshold for determining the similarity between two samples, namely,  $s_1$  and  $s_2$ , both belonging to the dataset of samples, which is defined as  $D$ . It generates a graph file in *dot* format [27] in which the nodes represent the executable files, and an edge between two nodes represents the fact that between them there is a similarity greater than the established threshold. The generation of the graphs is computationally expensive since it calculates the similarity for each different pair of samples.

#### 4. Experiments and Results

In this section, the experiments and results obtained using our malware analysis and clustering framework are presented.

**4.1. Overview.** In order to test the platform described in Section 3, we built different custom virtual machines using *buildroot* [28], which automates the process of building an embedded Linux system. In total, we built machines for the five most widely used architectures in the current IoT market, namely, Intel 80386, x86-64, MIPS, ARM, and PowerPC, generating a file system and a compilation of a kernel image for each one. We used *strace* as a monitoring tool to obtain the execution traces.

To perform the analysis, we used different samples of Linux-based malware which targets IoT devices. The samples are distributed among the five architectures mentioned. The malware samples are labeled using AVClass [29], which categorizes them using a ranking of the labels provided by different antivirus engines. Table 2 summarizes the number of pieces of malware used for each architecture and how many of them are packed and labeled.

Finally, we used our framework to analyze all the samples and visualize the relationships between them according to the metrics described in Section 3.4. The following sections show the results obtained after analyzing the entire set of samples described above in terms of static and dynamic points of view.

**4.2. Static.** In this section we present the results of the analysis and clustering processes using the static features described in Section 3. We use a threshold, which can be adjusted by the user, of 0.8 to determine whether two samples are related for both metrics. This value selection is based on an empirical study which is out of the scope of this paper.

**4.2.1.  $n$ -grams.** We use the  $n$ -grams of the operation codes extracted in the static analysis process. The size was empirically determined to be four by using cross validation. Since the operation codes are architecture dependent, we generated clusters for each of the architectures independently. Figure 3 shows the graphic for all architectures in the study, namely, MIPS, PowerPC, x64, x86, and ARM. The nodes represent malware samples and the edges indicate whether there is a

similarity greater than 0.8 at the  $n$ -gram level. Gray is used to represent malware samples that do not have a label and the rest of the colours represent each of the families that have been labeled (AVClass) in the dataset. As can be seen, there are different clusters formed mainly of samples from the same family. In some cases, there are related samples from several families. This may be because some of the samples are packed and, if they use the same packer, they may share the same code routines to unpack the executable at run time. One of the disadvantages of using static features is that they can be affected by code obfuscation. This metric can also be affected depending on whether the executable is compiled with static linking or with dynamic linking, since those binaries compiled with static linking could have more unique  $n$ -gram sequences because the functions imported from the libraries are included in the binary itself. In general terms, the proposed architecture detects well the families of malware samples for all the architectures.

**4.2.2. Cyclomatic Complexity.** We use cyclomatic complexity to cluster the samples. Since the metric is extracted from disassembled programs and depends on the assumptions of the compiler and the assembly code that it generates, we cluster the samples for each of the architectures independently. This is because, after looking at several executable files available for different architectures (e.g., *busybox*), we observe that the cyclomatic complexity for the same functions varies according to the architecture. Although it is not very different between one and the other, it does change even if they have been compiled with the same compilation options. Figure 4 shows the graph for all the architectures used in this paper. As we can see, the clusters generated belong to the same family, and there are several small clusters for the same family, such as *Gafgyt*, *Tsunami*, or *Mirai* for the ARM architecture. This is due to the fact that this metric measure similarity at a structural level between two samples. Therefore, it can also be affected by obfuscated code. In addition, if a sample is compiled in a static way and another in a dynamic way, there will not be a structural similarity between them (those compiled with static linking have imported library functions within the executable instead of being resolved at runtime as in binaries compiled with dynamic linking).

Observing the graphs generated for both metrics (Figures 3 and 4), it can be seen that, in general, the clusters created using  $n$ -grams are made up of more samples than those produced using cyclomatic complexity. In either case, most of the connected samples are related to others from their own family without producing many false positives.

**4.2.3. Dynamic.** In this section, we present the results obtained in the clustering process using the dynamic characteristics extracted in Section 3.5 and the metric described in the same section. As was done in Section 4.2.2, we use a threshold of 0.8 to match two malware samples. We use sequences of  $n$ -grams of size four for the *syscalls* executed for

TABLE 2: The number of malware samples distributed for each of the architecture.

Arch	Samples	Packed	Labeled
Intel 80386	279	58	211
X86-64	344	168	134
MIPS	318	63	288
ARM	246	24	200
PowerPC	275	12	258
	1462	325	1091



FIGURE 3: Continued.



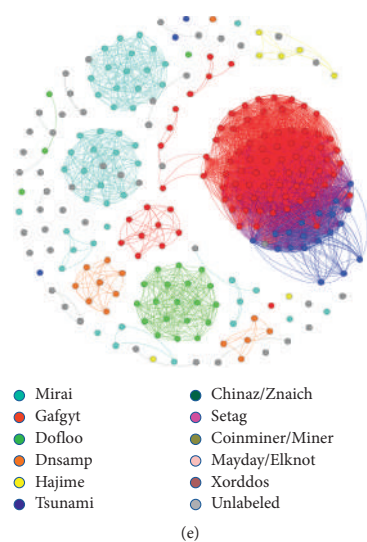


FIGURE 3: Clusters generated for the MIPS (a), PowerPC (b), x64 (c), x86 (d), and ARM (e) architectures using  $n$ -grams and the Jaccard index to calculate the similarity.

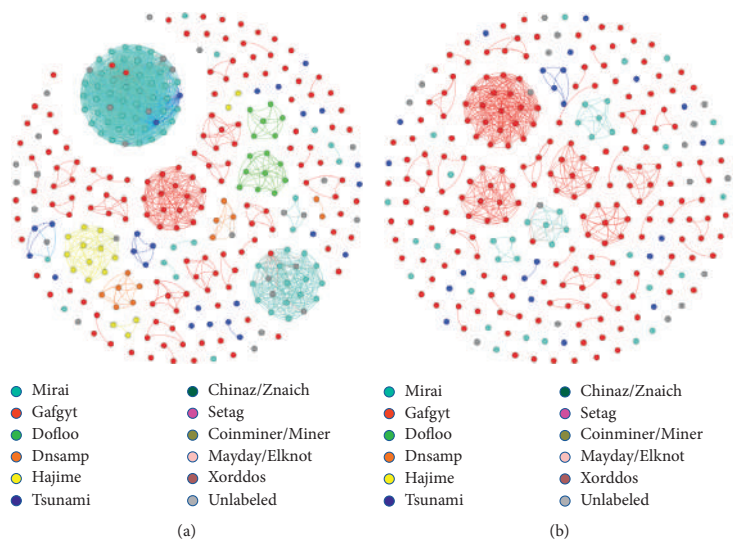


FIGURE 4: Continued.

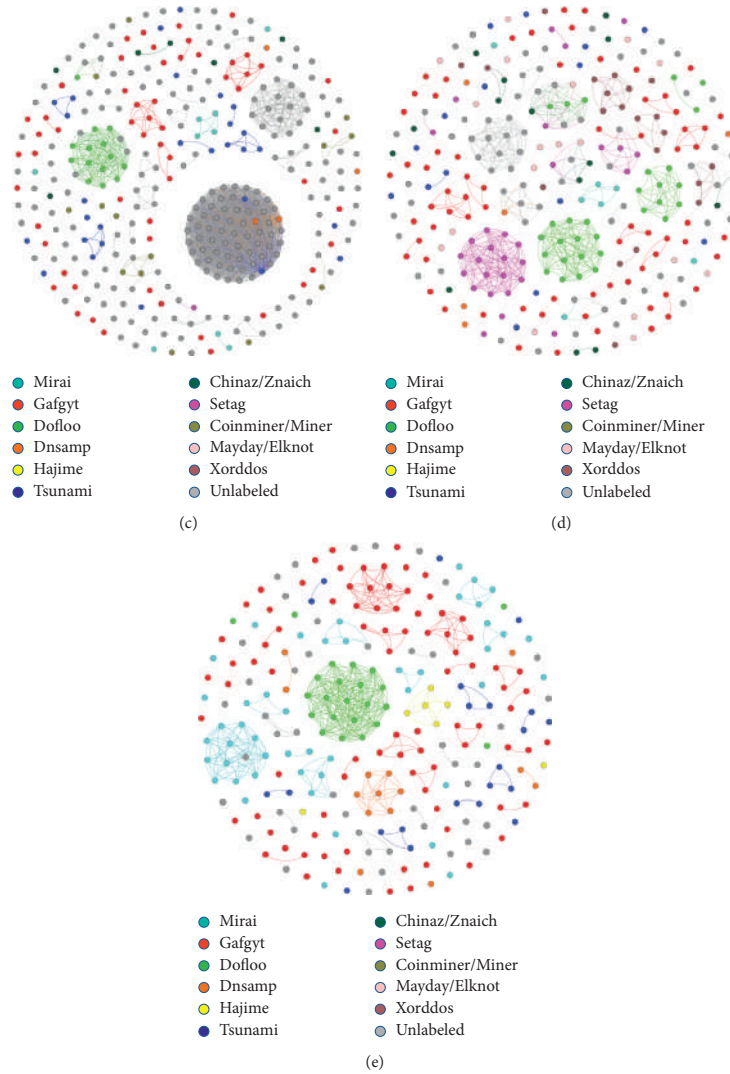


FIGURE 4: Clusters generated for the MIPS (a), PowerPC (b), x64 (c), x86 (d), and ARM (e) architectures using cyclomatic complexity and the custom function described in Section 3.

each of the samples. Since the *syscalls* are petitions to the operating system to request a service (e.g., create a socket and kill a process), and these have the same name in any Linux-based operating system, using them for clustering allows us to find similarities between the execution traces of samples from different architectures.

Figure 5 shows the clusters generated using the *syscalls* traces as features. On the left, each sample is colored depending on the architecture to which it belongs. On the right, each sample is colored depending on the family to

which they belong, with gray indicating the unlabelled ones. It can be observed that there are clusters that are formed of samples from different architectures, such as MIPS, PowerPC, and Intel 80386. If we observe these same clusters in the family-categorized image, it can be seen that the samples belong to a particular malware family. In addition, it can be noticed that the clusters are made up of samples from the same family, and that, based on their behavior, pieces of malware from different architectures have been categorized into the same cluster.

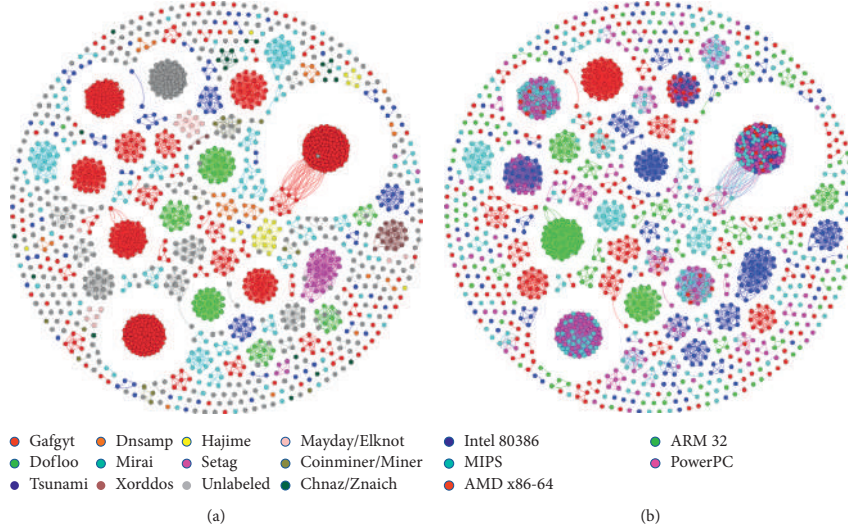


FIGURE 5: Clusters generated for all architectures using the execution traces obtained in the dynamic analysis. The  $n$ -gram size used for the *syscalls* sequence is four. The edges connect those samples with a similarity index greater than 0.8.

Finally, we observe that there are different clusters for the same family. Unlike the previous case, in which the samples may appear different depending on the architecture for which they were compiled or the different compilation options, now it may indicate that they belong to different campaigns of the same family. Malware is constantly evolving, and its creators add new functionalities or use existing ones from other pieces of malware that have proven effective and beneficial. Also, it should be noted that the original source code of some of the most widely used malware families is available on the Internet, such as *Gafgyt* or *Mirai* [18], and there may be variants created by different authors.

## 5. Conclusions

In this proposal, we have addressed IoT malware analysis, focusing on the automatization of the examining process. Our motivation for this is the huge increase in cyberattacks that have been carried out in this environment over recent years, which has led to the impossibility of manually studying the samples as the number is too immense. After evaluating the proposals from the community, it has been observed that there were none that focused on both analyzing (statically and dynamically) a large number of IoT malware samples at once and providing compatibility with several architectures.

Consequently, a multiarchitecture framework for automatic malware analysis and clustering has been presented. The proposal, which is based on a modular approach and supports samples from five different IoT architectures, namely, ARM, PowerPC, MIPS, Intel 8086, and x64-86, is able to extract static and dynamic features from a sample and compare it with previous analyzed ones, categorizing it into

families depending on the similarity. In addition, besides saving a considerable amount of time when examining pieces of malware, it offers flexibility to the user, allowing them to define their own emulated architectures and to adapt the threshold used to determine whether a sample is categorized into a family or not.

The proposal has been evaluated through the examination of nearly 1,500 malware samples from the five architectures that are supported by the framework, offering promising results and proving its effectiveness when clustering malware samples. Especially relevant is the outcome of the dynamic analysis, in which the proposal has been able to cluster samples from multiple malware campaigns, even if they were designed for different architectures. In addition, it has been detected that, when clustering using the static features, samples may appear different depending on the architecture for which they were compiled or the different compilation options. Other factors, such as code obfuscation, also hinder the task, although the results generated by the static analysis are also satisfactory.

Given the good results offered by the framework when tested and knowing the importance of improving the analysis of malware samples, there are several lines of research that could be followed to complement this proposal. Some such projects could be to

Study the network communications made by the malware samples when they are executed and use them as a feature to cluster them

Expand the visualization features, offering the user an interactive representation of the results, allowing them to directly browse through the different samples or filter them by selecting certain characteristics.



Add other IoT architectures so that samples designed for them could also be examined.

Employ other metrics to determine sample similarity, and even to use advanced machine learning techniques to add a layer of intelligence to the framework.

## Data Availability

The sample data used to support the findings of this study are available from the corresponding author upon request.

## Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

## Acknowledgments

This work was supported by the MINECO and European Commission (FEDER funds) under project RTI2018-098156-B-C52, the JCCM under the project SB-PLY/17-180501/-00035, the Spanish Education, Culture and Sports Ministry under grants FPU 17/03105 and FPU 17/02007, the University of Castilla-La Mancha under the contract 2018-PREDUCLM-7476 and the project 2020-GRIN-28846, and the Spanish State Research Agency under the project PEJ2018-003001-A.

## References

- [1] D. Demeter, M. Preuss, and Y. Shmelev, "IoT: a malware story-securelist," 2019.
- [2] The Council of Economic Advisers - United States of America and CEA Report, *The Cost of Malicious Cyber Activity to the U.S. Economy*, The Council of Economic Advisers, Washington, DC, USA, 2018.
- [3] E. L. Xua and L. Ling, "Industry 4.0: state of the art and future trends," *International Journal of Production Research*, vol. 56, pp. 2941-2962, 3 2018.
- [4] P. P. Gaikwad, J. P. Gabhane, and S. S. Golait, "A survey based on smart homes system using internet-of-things," in *proceedings of the 2015 International Conference on Computation of Power Energy, Information and Communication (ICCPEIC)*, Piscataway, NJ, USA, 2015.
- [5] Gartner Says 8.4 Billion Connected Things Will Be in Use in 2017 <https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016>.
- [6] Gartner Says 5.8 Billion Enterprise and Automotive IoT Endpoints Will Be in Use in 2020 <https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-iot>.
- [7] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, and B. A. Hamilton, "Reference model for service oriented architecture 1.0," *The Organization for the Advancement of Structured Information Standards*, vol. 12, 2006.
- [8] Y. M. P. Pa, S. Suzuki, K. Yoshioka et al., "IoT POT: a novel honeypot for revealing current IoT threats," *Journal of Information Processing*, vol. 24, no. 3, pp. 522-533, 2016.
- [9] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding linux malware," in *proceedings of the 2018 IEEE Symposium on Security and Privacy*, Security and Privacy, Francisco, CA, USA, July 2018.
- [10] A. Costin and J. Zaddach, "IoT malware: comprehensive survey," *Analysis Framework and Case Studies*, BlackHat, Las Vegas, NV, USA, 2018.
- [11] C. Guarnieri, "Cuckoo sandbox-automated malware analysis," 2016, <https://cuckoosandbox.org/>.
- [12] <https://github.com/monnappa22/Limon>.
- [13] [detux/detuxhttps://github.com/detux/detux](https://github.com/detux/detux).
- [14] K.-C. Chang, R. Tso, and M.-C. Tsai, "IoT sandbox: to analysis IoT malware zollard," in *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*, New York, NY, USA, September 2017.
- [15] T. N. Phu, K. H. Dang, D. N. Quoc, N. T. Dai, and N. N. Binh, "A novel framework to classify malware in mips architecture-based IoT devices," *Security and Communication Networks*, vol. 2019, Article ID 4073940, 13 pages, 2019.
- [16] M. Alhanahnah, Q. Lin, Q. Yan, N. Zhang, and Z. Chen, "Efficient signature generation for classifying cross-architecture IoT malware," in *Proceedings of the 2018 IEEE Conference on Communications and Network Security (CNS)*, Beijing, China, June 2018.
- [17] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, "Lightweight classification of IoT malware based on image recognition," in *Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Tokyo, Japan, July 2018.
- [18] R. Kumar, X. Zhang, R. U. Khan, and A. Sharif, "Research on data mining of permission-induced risk for android IoT devices," *Applied Sciences*, vol. 9, 2019.
- [19] T. Lei, Z. Qin, Z. Wang, Q. Li, and D. Ye, "EveDroid: event-aware android malware detection against model degrading for IoT devices," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6668-6680, 2019.
- [20] R. Team, *Radare2 Book*, 2017.
- [21] A. H. Watson, D. R. Wallace, and T. J. McCabe, "Structured testing: a testing methodology using the cyclomatic complexity metric," *US Department of Commerce, Technology Administration, National Institute of Standards and Technology*, vol. 500, 1996.
- [22] libvirt The virtualization API <https://libvirt.org/>.
- [23] Q. E. M. U. 2020, <https://www.qemu.org/>.
- [24] strace trace system calls/signals, <https://linux.die.net/man/1/strace>.
- [25] SystemTap, <https://sourceware.org/systemtap/>.
- [26] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*, Cambridge university press, Cambridge, UK, 2014.
- [27] Graphviz-Graph Visualization Software, <http://www.graphviz.org/documentation/>.
- [28] Buildroot-making embedded Linux easy, <https://buildroot.org/>.
- [29] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "A tool for massive malware labeling," in *Proceedings of the International Symposium on Research in Attacks Intrusions, and Defenses*, Paris, France, September 2016.



## CHAPTER 6

# Characterizing Linux-based malware: Findings and recent trends

- **Title:** Characterizing Linux-based malware: Findings and recent trends.
- **Authors:** J. Carrillo-Mondéjar, J.L Martínez and G. Suarez-Tangil.
- **Type:** Journal paper.
- **Journal:** Future Generation Computer Systems.
- **Publisher:** Elsevier.
- **ISSN:** 0167-739X.
- **Status:** Published.
- **Publication date:** September 2020.
- **Volume:** 110.
- **Paper Number:** 267-281.
- **DOI:** 10.1016/j.future.2020.04.031
- **JCR IF/ranking:** 7.187/Q1 (JCR2020).



Contents lists available at ScienceDirect

## Future Generation Computer Systems

journal homepage: [www.elsevier.com/locate/fgcs](http://www.elsevier.com/locate/fgcs)

## Characterizing Linux-based malware: Findings and recent trends

J. Carrillo-Mondéjar<sup>a,\*</sup>, J.L. Martínez<sup>a</sup>, G. Suarez-Tangil<sup>b</sup><sup>a</sup> Albacete Institute of Informatics (i3a), Universidad de Castilla-La Mancha, Albacete, Spain<sup>b</sup> Department of Informatics at King's College London (KCL), London, UK

## ARTICLE INFO

## Article history:

Received 23 September 2019

Received in revised form 11 April 2020

Accepted 19 April 2020

Available online 22 April 2020

## Keywords:

Malware forensics

IoT

Embedded systems

Data analytics

Machine learning

Expert systems

## ABSTRACT

Malware targeting interconnected infrastructures has surged in recent years. A major factor driving this phenomenon is the proliferation of large networks of poorly secured IoT devices. This is exacerbated by the commoditization of the malware development industry, in which tools can be readily obtained in specialized hacking forums or underground markets. However, despite the great interest in targeting this infrastructure, there is little understanding of what the main features of this type of malware are, or the motives of the criminals behind it, apart from the classic denial of service attacks. This is vital to modern malware forensics, where analyses are required to measure the trustworthiness of files collected at large during an investigation, but also to confront challenges posed by tech-savvy criminals (e.g., Trojan Horse Defense).

In this paper, we present a comprehensive characterization of Linux-based malware. Our study is tailored to IoT malware and it leverages automated techniques using both static and dynamic analysis to classify malware into related threats. By looking at the most representative dataset of Linux-based malware collected by the community to date, we are able to show that our system can accurately characterize known threats. As a key novelty, we use our system to investigate a number of threats unknown to the community. We do this in two steps. First, we identify known patterns within an unlabeled dataset using a classifier trained with the labeled dataset. Second, we combine our features with a custom distance function to discover new threats by clustering together similar samples. We further study each of the unknown clusters by using state-of-the-art reverse engineering and forensic techniques and our expertise as malware analysts. We provide an in-depth analysis of what the most recent unknown trends are through a number of case studies. Among other findings, we observe that: i) crypto-mining malware is permeating the IoT infrastructure, ii) the level of sophistication is increasing, and iii) there is a rapid proliferation of new variants with minimal investment in infrastructure.

© 2020 Elsevier B.V. All rights reserved.

## 1. Introduction

The security community has been fighting against malicious software (malware) for over three decades. Prior to the proliferation of smartphones, malware developers mainly targeted Windows due to its market share. More recently, the focus of malware authors has slowly but steadily shifted towards Linux-based operating systems, the main factor being its growing popularity and the huge number of computing devices that are part of the Internet of Things (IoT).

The IoT is globally expanding (it is expected there will be 50 billion devices by 2020 [1]), providing diverse benefits in nearly every aspect of our lives, such as industry, transportation, smart-cities, health-care, entertainment and our daily life. One of

the main characteristics of an IoT infrastructure is the ability to capture data, and this collected information can be the objective of attackers. Also, the huge number of devices with very limited characteristics in terms of available memory, bandwidth, power consumption, vulnerable software, etc. make it possible for a simple attack to achieve many objectives, for example to create a botnet network.

Embedded devices rely on a variety of different architectures. While PCs run predominantly on x86-flavored architectures, the IoT framework normally runs on open hardware architectures such as Advanced RISC Machine (ARM), PowerPC (PPC), and Microprocessor without Interlocked Pipeline Stages (MIPS), among others. From an attacker's point of view, IoT devices are quite attractive compared with PCs or laptops as they are online at all times and have no AntiVirus (AV) or Intrusion Detection Systems (IDS). These devices are created with innovative features to attract users, but often lack security and privacy measures. This makes many IoT devices vulnerable to critical security issues, ranging from the use of default passwords or insecure default

\* Corresponding author.

E-mail addresses: [javier.carrillo@uclm.es](mailto:javier.carrillo@uclm.es) (J. Carrillo-Mondéjar), [jose.luis.martinez@uclm.es](mailto:jose.luis.martinez@uclm.es) (J.L. Martínez), [guillermo.suarez-tangil@kcl.ac.uk](mailto:guillermo.suarez-tangil@kcl.ac.uk) (G. Suarez-Tangil).<https://doi.org/10.1016/j.future.2020.04.031>

0167-739X/© 2020 Elsevier B.V. All rights reserved.

settings to outdated software with known security issues [2]. All these weaknesses give attackers powerful attack vectors.

The trend for Linux-based malware that is especially designed for the IoT is relatively new compared with classical malware, and therefore the number of known malware families is still small. The first malware family especially designed for IoT devices is the Mirai botnet. This malware family was mostly aimed at performing Distributed Denial of Service (DDoS) attacks. The source-code was released in 2016 but this malware was first seen at the end of 2014. Since the release of Mirai's source-code, new variants of malware have proliferated. While some of these variants exhibit similar behaviors to that of Mirai,<sup>1</sup> others present novel features and display completely different behaviors. Despite recent commercial reports having shown that IoT malware has been constantly growing [3], there is very little understanding of what the main characteristic features of recent malware samples are.

Due to the coexistence of multiple architectures in the IoT, malware analysis is an important part of the forensic process,<sup>2</sup> allowing the forensic analyst to know how a sample behaves and what parts of the system it modifies. Therefore, machine learning and pattern recognition can be seen as a new paradigm for digital forensics [4,5]. This is particularly relevant in malware forensics, where human analysts are confronted with a challenging adversarial task [6]. Advances in digital forensics require automated processes to aid malware analysts in the process of understanding: (i) whether a binary seen in an investigation is malware or not (malware detection), and (ii) which type of malware it is and what its expected behavior (malware characterization) might be. Furthermore, machine learning can be used to measure the trustworthiness of files collected during an investigation, but also to tackle challenges posed by tech-savvy criminals, such as the Trojan Horse Defense [7].

**Contributions.** This paper presents a comprehensive study of malware that is especially designed for Linux-based systems and tailored to IoT embedded devices. Our main contributions are as follows. (i) We build an automated system to study the problem of malware in IoT systems. Our system uses both static and dynamic analysis, together with a similarity function designed to relate unknown samples to known threats. (ii) We use our system to assist a malware expert (one of the co-authors of this paper) in the process of vetting recent unknown IoT malware samples and characterizing their behavior. We then present an in-depth analysis of the most recent unknown trends through a number of case studies. (iii) We dissociate the problem of understanding Linux-based IoT malware from the IoT architecture used for malware in the wild. We show the influence that the architecture has on the spread of malware. Finally, we have released the characterization produced for each of the clusters.<sup>3</sup>

**Findings.** By systematizing the analysis of the most representative dataset of Linux-based malware collected by the community to date [2] and dissecting samples in the most relevant clusters, we have made the following findings:

1. We show that malware designed for architectures such as x86\_64 is quite different to that designed to attack dedicated IoT architectures. Moreover, we also show that applying Machine Learning (ML) to understand large datasets of samples is challenging, especially when there is "noise" in the dataset. Due to the fact that malware especially

designed for Linux-based systems is still simpler, the techniques based on static analysis work well, mainly because the disassembly code of malware is not obfuscated. However, dynamic analysis generally fails for two reasons: (i) many samples cannot be executed on custom emulated IoT devices, as they do not provide the correct environment for which the malware has been designed; and (ii) the Command and Control (C&C) server of the sample changes quickly over time, rendering the sample inoperative.

2. The level of sophistication of Linux-based malware varies significantly. It ranges from malware which uses classical but effective techniques such as brute force attacks, to malware which exploits some vulnerability to spread to other victims. Here, we show that crypto-mining malware is currently targeting IoT platforms. Finally, malware designed for the creation of bots to carry out denial of service attacks continues to appear.
3. Linux-based malware is publicly available on the Internet, causing the rapid proliferation of new variants which base their main techniques on artifacts that are publicly available on the Internet (e.g., on GitHub).

The rest of the paper is organized as follows. Section 2 introduces the threat overview related to this work. In Section 3.1 the methodology for understanding IoT malware is addressed. Section 4 presents the results of the classifiers when applied to labeled samples and the visualization of data. The results of the classifier tests with unlabeled samples are shown in Section 5. Section 6 shows the analysis of different groups of unlabeled samples related to each other. Section 7 contains a discussion about IoT malware, taking as its starting point the analysis presented in this paper. The state of the art in malware especially designed for IoT platforms is described in Section 8. Lastly, our conclusions are presented in Section 9.

## 2. Threat overview

The IoT is a quickly developing entity made up of embedded and multi-platform devices. The majority of IoT malware focuses on devices with default login credentials, and these devices are being targeted by newly discovered Linux malware. In the last few years, malware in the IoT has gained more attention due to the damage caused by large-scale attacks such as the one carried out by the Mirai botnet. The increase in IoT devices together with the lack of updates in the face of the emergence of new vulnerabilities has led to the proliferation of malware targeting these platforms. In addition, the publication or leaking of the source code of some families such as Mirai or Bashlite causes new variants to appear or the creation of new families using part of their functionality or method to spread and infect new devices.

Unlike conventional malware that is aimed at a platform and/or specific architecture, IoT malware is available for different architectures such as MIPS, ARM or PowerPC, among others. Due to the wide range of Linux-based IoT devices, the creation of malware capable of attacking and infecting many of them has become one of the priorities of cybercriminals.

Currently, the methods used to spread through the network are not very sophisticated. Primarily, they rely on the use of brute force attacks using default credentials or known vulnerabilities whose exploits are in most cases published on the Internet. In this way, they take advantage of the lack of updates and the use of default device settings by users.

Initially, IoT malware focused on the creation of bots to be marketed for the carrying out of denial of service attacks. Nowadays, the trend is changing and new samples are appearing for other purposes such as mining, which tries to take advantage

<sup>1</sup> In fact, these variants are adaptations of Mirai source code.

<sup>2</sup> Locate, identify, collect and acquire data which may be relevant to an investigation.

<sup>3</sup> Characterization of the clusters can be downloaded from here: <https://bitbucket.org/Dankitan/characterizing-linux-based-malware-findings-and-recent-trends/src/master/>.

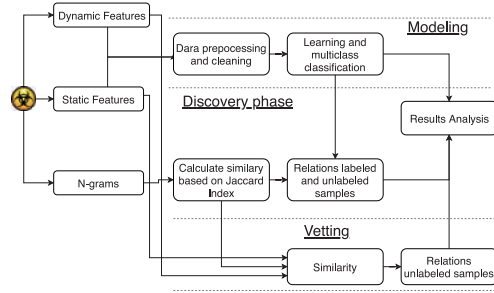


Fig. 1. Overview of our system.

of existing resources in the form of infected devices to mine cryptocurrencies.

Due to the time required to perform a manual malware analysis and the large number of attacks that appear every day, the security community is focusing its efforts on fighting a threat that is constantly growing. Therefore, it is necessary to build automated systems that, in addition to detecting new samples, allow the extraction of knowledge about the malware as well as its classification.

### 3. Methodology

In this section we present our methodology. We first provide an overview of the system and then describe each step in our pipeline.

#### 3.1. Overview

Fig. 1 shows an overview of the main building blocks of our pipeline, which consists of three phases. First, we have the feature extraction and the Modeling phase. The idea of this phase is to perform an exploratory analysis to guide the feature engineering process. Once the most relevant features in the IoT realm are selected, we build a supervised ML classifier capable of characterizing different known threats. Second, there is the Discovery phase. In this phase, we use the classifier trained in the previous phase and a metric based on sequences of opcodes from the disassembling of the malware to relate unknown samples to samples of known threats. Finally, we have the Vetting phase, in which we unveil novel threats that are unknown to the community. Specifically, we combine the features engineered in the first phase and the metric computed in the second phase with a custom distance function. This distance function is used to perform an unsupervised clustering task on unknown threats.

#### 3.2. Modeling

In this section, we present the details of the Modeling phase. This phase is responsible for the collection and cleaning of the data as well as the selection of the characteristics that will be used for the learning of ML algorithms. The results of this phase will be presented in Section 4.

##### Data acquisition and cleaning

This research is based on a sizable dataset of Linux-based malware samples. Linux is the main operating system for the IoT infrastructure. For each malware sample, the dataset contains: a

**Table 1**  
Samples distributed by architectures.

Architecture	Labeled	Not labeled
ARM 32-bit	493	62
ARM 64-bit	31	16
AMD x86-64	1765	1253
Intel 80386	607	113
MIPS I	1960	160
Motorola 68000	1169	47
SPARC	1144	26
SPARC Version 9	0	1
PowerPC	1509	60
PowerPC 64-bit	1	0
Hitachi SH	127	3
EM_ARC_A5	1	0
	8807	1741

binary, features extracted from an automatic analysis and, in the majority of cases, a label characterizing the type of malware using AVClass [8] for readily available threat intelligence. Thus, the dataset consists of a set of different malware samples distributed among different architectures with Linux as the target operating system. The dataset used in this paper has recently been collected by the authors in [2] and the threat intelligence has been obtained from VirusTotal [9]. As mentioned above, AVClass has been used for labeling the samples. It is a tool which unifies sparse labels given by different AVs to each malware sample by assigning them to a malware family. Typically, AVClass relies on VirusTotal reports to label each sample on the basis of the labels provided by the AV. Basically, AVClass receives the labels from the AV engines and, once the duplicated labels have been removed, a tokenization and filtering process is applied. Finally, it applies an alias detection process and a ranking, using the most frequent labels as the family name for the family of a sample. One of the main limitations of AVClass is that it cannot label samples when there is no consensus among a minimum number of AV engines. Therefore, the final decision of labeling a malware sample depends on the quality of the AV labeling process. This process is an error-prone task, as pointed out previously [10] and as discussed below in Sections 4 and 6.3.

After obtaining the data and the labels, we clean the dataset by removing those samples that cannot be labeled by AVClass, either because they are not detected by more than two AV engines or because they are detected by the AV heuristics and, therefore, their labels are generic. Table 1 shows the number of samples distributed among the different architectures and how many of them are labeled.

##### Feature analysis

The first approach to dealing with the problem of understanding malware behavior, or even family malware classification, in the IoT is to apply Machine Learning techniques to as much information extracted from the dataset as possible. Although the use of ML to classify malware has been widely studied for other operating systems such as Windows or Android, there are not so many studies focused on different operating systems for the IoT such as Linux-based ones. One of the problems that we face when building a generalized model is the diversity of existing architectures, as Linux-based operating systems and compilers make the same sample look different even when it belongs to the same malware family. At this point, the first step was to try to determine the most significant features from among all the information available in the dataset. For this, we use the following feature extraction techniques.

- *Static features*: analysis focusing on intrinsic characteristics in the executable itself without actually executing it. Within this group are characteristics of the binary file such as headers, strings that appear in the executable, machine code, imports of the library functions, entropy, etc.
- *Dynamic features*: analysis focusing on features obtained through monitoring the execution of an executable. For example, system calls made to interact with the system, information about opened files, memory, etc, that is, information about the behavior of the executable at run time.
- *Hybrid features*: the combination of the two previous methods, so that the characteristics used are extracted in a static and dynamic way.

To design our model, we decided to follow the three approaches. Although using only static features can cause a bad generalization of the model, since some of these characteristics could change between different architectures (especially in the IoT scenario), we believe that there are some static characteristics that should remain unchanged between different platforms, as for example strings. In terms of dynamic features, the behavior of the same sample must be similar regardless of the architecture on which it is executed. For this reason, we started with a single static model, then we included a dynamic model in the analysis and, finally, a model that combines both characteristics has been developed.

## Static features selected

Machine learning models require a diverse set of features to learn. One of the disadvantages of static features is that they are susceptible to obfuscation, although in the IoT scenario it is not very common, with around only 3% of the samples inside the dataset being packed, most of them using Ultimate Packer for Executable (UPX) or some variant of it. This is mainly because, nowadays, malware for Linux-based systems does not implement techniques to avoid AV or IDS; with resource-constrained devices, such as those used in the IoT infrastructure, there is not enough memory or battery power to deploy them. Thus, the proposed algorithm firstly looks for static features that do not change too much between different platforms. We do not consider features such as assembly code to be a good option since the same sample can be compiled for different architectures, and the instructions set for each one is different. This is an important issue because, as was shown in Table 1, the number of different architectures for Linux-based operating systems is large. Although there are several static features that could work well, such as text strings, we have focused on data based on Cyclomatic Complexity. Cyclomatic complexity [11] is a metric that is used in software engineering to calculate, in a quantitative way, the complexity at the logical level of a program or function. In our case, we have used the Cyclomatic Complexity of all program functions, as well as the maximum and average complexity. For example, say a malware sample has five functions with Cyclomatic Complexities 5, 7, 5, 3 and 5 respectively. We then account for 3 functions with complexity 5, 1 with 7 and 1 with 3. The maximum would be 7 and the average 5. We also use the number of basic blocks and its average. A basic block consists of a set of instructions that begins with a single instruction and ends with an instruction that redirects the execution flow to another basic block or ends the program. These characteristics allow us to know whether two programs are structurally similar at the machine code level. Since there are very few packed samples and we wish to check whether the use of static features works well for IoT malware, we have discarded packed samples from the dataset.

**Table 2**

Number of final samples.

Architecture	Labeled	Not labeled
ARM 32-bit	445	47
MIPS I	1705	63
AMD x86-64	1600	787
Intel 80386	523	71
PowerPC	1492	49
TOTAL	5765	1017

## Dynamic features selected

Regarding characteristics based on behavior which are collected at run time, we have chosen the following features:

- Unique syscalls.
- Ioctl.
- Rename process.
- Number of processes created.
- Check user and group identifier.

These features have mainly been encoded in a binary way (one or zero), indicating whether they are used (one) or not (zero). For example, if a process uses the ioctl SIOCGIFHWADDR command to obtain the MAC address of an interface, we use the number one to indicate that sample uses that feature. Otherwise, it will contain a zero. The number of processes created is encoded as a numerical feature.

While dynamic features are extremely useful to characterize the behavior of a program, we note that it is not always possible to obtain features for all samples. This is because performing dynamic analysis for certain IoT architectures is a challenging task [2]. This is due to the problem of managing different virtual machines to emulate a wide range of architectures. In addition, analyzing malware is a time sensitive task as, at the time of execution, C&C infrastructures or other external resources needed for the execution of the malware might be unavailable.

Moreover, families that do not contain at least ten samples have been removed from the dataset because we consider that less than ten samples is not enough for the model to generalize correctly. Table 2 shows the final sample set once the families are extracted. Likewise, packed samples and samples that do not contain dynamic feature are also filtered out from our analysis.

## 3.3. Discovery phase

This phase is aimed at characterizing the dataset of samples that are not labeled. For this, we will use the classifier built in the previous phase. Note that it is not possible to assess quantitatively whether the predictions made by the classifier are correct or not (there are no labels). Thus, in this phase we measure the similarity between predicted samples and labeled samples as a method to evaluate our accuracy (Section 5). Intuitively, samples that are associated with a family might be structurally similar to the samples of that family. This holds true for samples that are not obfuscated and thus we see this evaluation criteria as an under-approximation. It is also important to note that the performance of the classifier is formally evaluated in the previous phase using k-fold cross-validation, as shown in Section 4.1. However, this phase aims at discovering unknown threats and the similarity metric should be seen as a means of providing confidence in the decisions of the classifier.

The similarity metric is calculated as follows. First, we disassemble each executable. Then, we compute the sequences of operation codes (opcodes) of size  $n$ , namely  $n$ -grams. Listing 1 shows a small example with five assembly code instructions. For a sequence of  $n = 3$ , the resulting set of three-grams is: (push, mov, push), (mov, push, push) and (push, push, sub).



```

1 push rbp
2 mov rbp, rsp
3 push r12
4 push rbx
5 sub rsp, 10h

```

Listing 1: Extracting n-grams

Therefore, given two sets of n-grams, the similarity between them can be calculated using the Jaccard index [12] as:

$$jacc(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|}, \quad (1)$$

where the numerator indicates the number of subsets (denoted by  $|s_i|$  for  $i \in \{1, 2\}$ ) that are found in both samples,  $s_1$  and  $s_2$ , and the denominator indicates the total number of unique subsets between  $s_1$  and  $s_2$ . The result is a value between 0 and 1 that indicates the degree of similarity between two sets. As mentioned above, the similarity based on n-grams works well as long as the samples are not obfuscated or packed. The results for the number of unlabeled samples that match at the n-gram level with labeled samples, and the number of hits of the classifier, are presented in Section 5.2.

### 3.4. Vetting

While the previous phase aims at associating unlabeled samples with known threats, this phase aims at characterizing novel threats from our dataset of non-labeled malware. For this, we cluster samples in the non-labeled set using unsupervised learning. This clustering process is driven by a custom distance function that relies on both the features extracted in phase one (Section 3.2) and the similarity index from phase two (Section 3.3). This function is formalized as follows:

$$distance(s_1, s_2) = \frac{\sum_{i=0}^{|F|} \frac{\min(f_i^{s_1}, f_i^{s_2})}{\max(f_i^{s_1}, f_i^{s_2})} \times \frac{1}{|F|} + jacc(s_1, s_2)}{2} \quad (2)$$

where  $f_i^{s_1}$  is the feature  $i$  of sample  $s_1$  (likewise for  $s_2$ ) and  $|F|$  is the total number of features. In other words, we normalize the feature vectors with respect to all other samples in our dataset samples. Given two vectors, the similarity index is calculated by means of the average distance of each pair of features. For example, given a sample with 100 functions and the use of the read, write and execve syscalls, and another sample with 80 functions and the use of the read, write and open syscalls, we obtain the following two vectors: (100, 1, 1, 0, 1) and (80, 1, 1, 1, 0). The resulting similarity index is 0.56, which is computed as follows:  $((0.8/1 + 1 + 1 + 0 + 0)/5)$ . This means that the similarity between the two sets of features is 56%. Then, we compute the average between the index of similarity obtained at the n-gram level (calculated in the previous section) and at the level of features (Section 3.2).

To cluster samples together, we use the similarity function in Eq. (2). We consider that two samples are in the same cluster if they have an index higher than 0.8, that is, if they have a ratio greater than 80%. We chose this empirically after testing different thresholds. A higher threshold does not produce a notable increase in performance. However, a lower threshold produces false positives. We also use this distance function to represent the relationships between the malware samples in the form of a graph. For the purpose of our study, we consider that clusters of unlabeled samples alone constitute a group of samples that belong to a threat or family that it is currently unknown. Where applicable, we also connect clusters of unlabeled samples to clusters of known threats using the threshold described above. This enables us to vet the discovery phase discussed above (see Section 3.3). The most interesting clusters are further discussed in Section 6.

Table 3

Results for the different types of algorithms and characteristics used in our model.

Feature type	Algorithm	Features	Precision	Recall	FScore
Static	K Neighbors	470	80.99	78.22	77.27
		220	80.67	78.42	77.38
	SVM kernel='rbf'	470	73.46	80.91	72.53
		220	74.23	82.29	73.29
	SVM kernel='Linear'	470	62.10	56.33	57.09
		220	58.10	54.64	53.71
Dynamic	Decision tree	470	82.18	83.44	79.61
		220	80.58	83.18	78.85
	Random forest	470	<b>83.01</b>	<b>83.67</b>	<b>81.02</b>
		220	<b>84.23</b>	<b>84.41</b>	<b>81.70</b>
	K neighbors	211	<b>86.17</b>	82.08	82.46
		134.67	<b>84.79</b>	81.42	81.99
Hybrid	SVM kernel='rbf'	211	83.29	85.58	82.38
		134.67	82.28	86.50	82.13
	SVM kernel='Linear'	211	84.49	82.38	81.70
		134.67	82.09	81.15	80.32
	Decision tree	211	80.06	86.78	81.26
		134.67	80.32	88.71	81.93
Hybrid	Random forest	211	83.28	<b>87.60</b>	<b>84.08</b>
		134.67	82.72	<b>88.96</b>	<b>83.91</b>
Hybrid	K neighbors	681	86.60	83.40	83.31
		269.5	86.31	84.43	84.04
	SVM kernel='rbf'	681	<b>88.80</b>	84.17	84.47
		269.5	<b>88.26</b>	86.07	85.35
	SVM kernel='Linear'	681	87.92	85.39	85.37
		269.5	83.14	81.47	81.02
Hybrid	Decision tree	681	81.36	84.80	80.30
		269.5	82.31	83.53	80.57
Hybrid	Random forest	681	88.08	<b>87.17</b>	<b>86.55</b>
		269.5	87.70	<b>86.61</b>	<b>86.07</b>

## 4. Modeling

This section evaluates the effectiveness of the modeling phase presented in Section 3.2.

### 4.1. Machine learning

First, static features have been used to train the model. Then, dynamic characteristics have also been used and, finally, all the features have been included. Basically, we have carried out these experiments with different types of characteristics to understand how much each characteristic contributes to machine learning algorithms. This is because the automatic extraction of static features is far more efficient than automatic dynamic analysis, which in the best case can take several minutes of execution in a sandbox environment, causing the extraction of features to consume a lot of time.

Then, these same tests have been performed by applying feature selection to eliminate those features that contribute least to machine learning algorithms. Different machine learning algorithms have been used, such as Random Forest [13,14], K-nearest [14,15], Decision Tree [14,16] and Support Vector Machines (SVM) [14,17] using the Radial Basis Function (RBF) and Linear kernels [14]. The implementation of these algorithms was provided by the sklearn library [18], which is written in python.

As a dataset, 5765 samples containing more than 20 different families have been used. K-fold [19] has been performed as a cross-validation strategy with the dataset being divided into 6 stratified folds, so that there is the same percentage of samples from each family in each split. Feature selection has been performed for each training split in the cross validation loop, since the test is considered unseen.



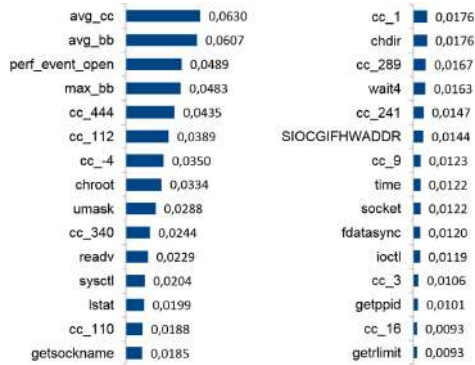


Fig. 2. Importance of the 30 features that contribute most value to the Random Forest algorithm.

As evaluation metrics, *precision*, *recall* and *Fscore* [20] have been used. These metrics have been applied to each class independently, with the final result being the average of all the classes, that is, each class has the same weight in the results regardless of the number of samples. Table 3 shows the results obtained with each of the classifiers. The best results obtained by the ML algorithms are in bold. It can be observed that in no case is there much difference in the use of all the features from the use of a subset of them, and in some cases it even improves the results when applying feature selection. In general terms, Random Forest offers the most balanced results in terms of *precision*, *recall* and *Fscore*. In addition, it can be observed that the use of only static features does not work very well with SVM.

As mentioned above, the selection of variables produces similar results in most cases to those when working with the complete set of characteristics. Fig. 2 shows the 30 most representative variables of the random forest (dynamic + static) algorithm, that is, those whose influence on the model is greatest. We can observe that the two characteristics that most influence this algorithm are the average cyclomatic complexity and the number of basic blocks. Among the other static characteristics that appear, it is important to highlight certain cyclomatic complexities of functions that have a very high value, such as cc\_444, cc\_340 and cc\_289. Among the features based on its behavior are those related to syscall sockets, ioctl, perf\_event\_open, chroot, etc.

Table 3 shows the results obtained for the different algorithms and features used. We can observe that most of the models yield a performance of around 85% on average. As we discussed above, the metrics used have been applied to each class independently. The overall results are summarized using the average, which does not take into account the size of the class. Here, we observe misclassifications of families with a small number of samples (e.g., 10 samples), which brings the average down despite having large classes (e.g., 2K samples) correctly classified. The confusion matrix in Fig. 3 details the percentage of correct and incorrect labels predicted in each class and provides a more accurate picture of the classification performance. In particular, as a representative example, Fig. 3 shows the confusion matrix generated for the Random Forest hybrid model. If we look this matrix, we can see that 25% of the time the chinaz family is treated as znaich, and 33% of znaich is treated as chinaz. Actually, this makes sense if we look at it: the name of the family is very similar, chi-na-z and z-na-ich, and therefore it is very likely that it is the same family which is labeled differently or that both families have

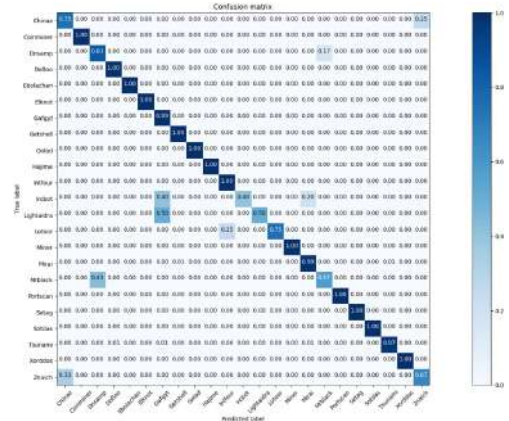


Fig. 3. Confusion Matrix of the Random Forest Algorithm using the hybrid approach.

been created by the same authors. The same case occurs with Mrblack and Dnsamp: several anti-viruses label these samples as Mrblack and others as Dnsamp, so both are the same family. In the case of Ircbot and Lightaidra, some samples could have been incorrectly labeled. After reviewing the VirusTotal reports of some samples labeled as Ircbot and Lightaidra, we observed that some anti-viruses mark some of them as Gafgyt or Mirai. This makes us think that the prediction could be correct and some of these samples are these kinds of malware, or that as many of the malware samples in the IoT have had their source code leaked, there are similar code pieces in samples from different families, causing the antivirus to label them incorrectly due to the fact that they can match with static signatures or on the basis of their heuristics. We should bear in mind that AVClass labels are determined by the consensus of the different labels that the AV gives to a sample. Therefore, the label selected as AVClass may be incorrect.

#### 4.2. Data visualization

The objective of this section is to visualize the set of samples using a three-dimensional plane so that we can observe in a more visual way whether the chosen characteristics allow the generalization of the families that are part of the dataset. Since the number of features is much higher, we have used T-Distributed Stochastic Neighbor Embedding (T-SNE) [21] to reduce the data dimensionality and visualize them correctly. In order to improve the visualization of the samples and make this clearer, we will only show the families with the largest number of samples on the chart.

Fig. 4 shows the most populated families of the dataset. We plot the families and their labels together with the unlabeled samples. Note that it includes samples compiled for different architectures. We see that our set of features are representative regardless of the architecture. In general, samples of a family are close in the plane to other samples of the same family. For example, the families with the largest number of samples. We observe that samples from Gafgyt, Mirai and Tsunami are grouped together. We observe also that there are sub-clusters which are attributed to either set of samples from the same architecture or to variants within the family. Intuitively, when we look at

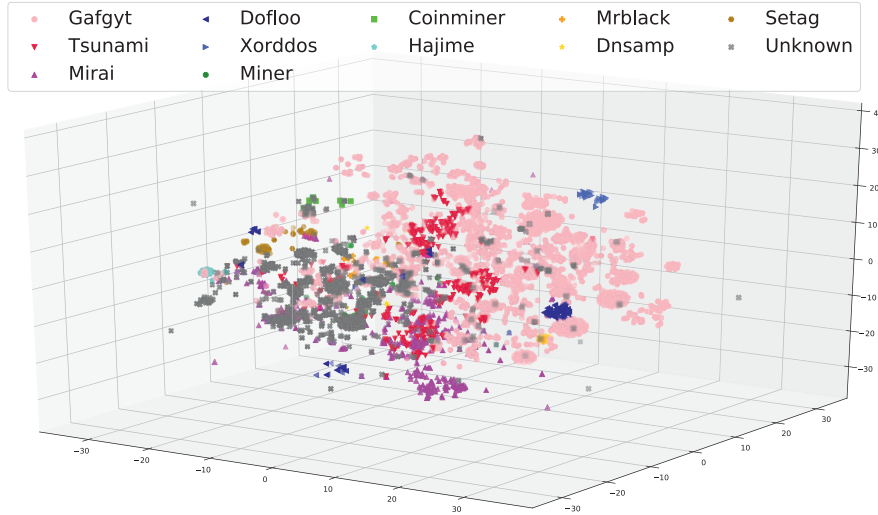


Fig. 4. Visualization of the labeled malware samples with the greatest presence in the dataset and unlabeled samples.

families that are less populated such as Hajime or Dofloo, we observe fewer sub-clusters. On the other hand, when we look at the representation of the unknown families, we see that they are very close to each other and distributed over several clusters. This tells us that unlabeled samples can be attributed to a handful of new, but prolific, families.

## 5. Discovery phase

In this section, we test the classifier with the unknown set of samples.

### 5.1. Description

As we mentioned in Section 3.3, this set of samples is not labeled so we do not have a quantitative way to know whether the prediction is correct or not. Therefore, we calculate the level of similarity between unlabeled and labeled samples and we check how many unlabeled samples are similar at the  $n$ -gram level with labeled samples and whether the label predicted by the classifier is correct in those samples that display similarity.

The steps followed to calculate the similarity are the following:

1. Extract the sequences of opcodes from each of the malware samples and generate the  $n$ -grams.
2. Compare each sample of unlabeled malware with the other samples of that same architecture since each architecture has a different set of instructions.
3. Check how many unlabeled samples are related to labeled samples.

The  $n$ -gram tests have been carried out evaluating different values of  $n$  (6, 8, 10, 12 and 14). In the case of a small size of  $n$ -gram, a high similarity level might be obtained due to the fact of having sets with small opcode sequences in different parts of the program functions, and the fact that the Jaccard Index only takes into account unique sets of  $n$ -grams. Increasing the  $n$ -gram size allows us to match similar samples in a more reliable way because it is more complicated to match sequences of, for example,

14 opcodes without there being any relation. The threshold used to consider that they are the same sample is 0.8, that is, that the degree of similarity between two samples is greater than 80%. As with the size of the  $n$ -gram, the higher the threshold, the greater the probability that two samples that exceed that threshold will be from the same family, that is, the probability of false positives will be lower. Based on the similarity at the  $n$ -gram level, there are between 126 and 134 (around 12%–13%) unlabeled samples that are related to samples whose label is available.

### 5.2. Evaluation

Once the similarities between unlabeled samples and others have been obtained, we can test the classifier with those samples whose label we do not know and verify, in a quantitative way, how many labels it correctly predicted. The procedure that has been carried out is the following:

1. We pass the new samples through the different classifiers using both static and dynamic features and feature selection, obtaining the predicted label for each of the samples according to the classifiers.
2. For each predicted sample we search with the samples with similarity higher than 80% calculated previously. We check those samples that have a label if it matches the label that the classifier predicts. A threshold of 80% has been chosen so that the samples that exceed it have a high probability of being from the same family.

Table 4 shows the results obtained for each algorithm and size of  $n$ -gram used. The results are classified by architecture, indicating how many samples have been predicted from the totals for that architecture. Also, the hits column is shown as the total number of hits with respect to the total number of unlabeled samples that have a relationship with samples with labels at that  $n$ -gram size. Finally, the last column shows the calculated percentage of success.

If we look at the table in more detail, we can see that with the ARM, MIPS and PowerPC architectures, almost all the algorithms come close to correctly predicting almost all the samples.

**Table 4**

Predictions of the classifiers with the unlabeled samples that maintain similarities at the n-gram level with labeled samples.

Algorithm	ngram	ARM 32-bit	MIPS I	AMD x86-64	Intel 80386	PowerPC	Hits	Percent
K neighbors	6	35/38	18/19	38/60	3/4	11/13	105/134	78,36%
	8	35/37	18/18	38/59	3/4	11/13	105/131	80,15%
	10	35/37	18/18	36/57	3/3	11/13	103/128	80,47%
	12	35/37	18/18	36/57	3/3	10/12	102/127	80,31%
	14	35/37	18/18	36/56	3/3	10/12	102/126	80,95%
SVM kernel=rbf	6	38/38	19/19	40/60	3/4	12/13	112/134	83,58%
	8	37/37	18/18	40/59	3/4	12/13	110/131	83,97%
	10	37/37	18/18	38/57	3/3	12/13	108/128	84,38%
	12	37/37	18/18	38/57	3/3	11/12	107/127	84,25%
	14	37/37	18/18	38/56	3/3	11/12	107/126	84,92%
SVM kernel=linear	6	37/38	19/19	39/60	3/4	12/13	110/134	82,09%
	8	37/37	18/18	38/59	3/4	12/13	108/131	82,44%
	10	37/37	18/18	36/57	3/3	12/13	106/128	82,81%
	12	37/37	18/18	36/57	3/3	11/12	105/127	82,68%
	14	37/37	18/18	36/56	3/3	11/12	105/126	83,33%
Decision tree	6	37/38	19/19	37/60	3/4	11/13	107/134	79,85%
	8	36/37	18/18	37/59	3/4	11/13	105/131	80,15%
	10	36/37	18/18	37/57	3/3	11/13	105/128	82,03%
	12	36/37	18/18	37/57	3/3	10/12	104/127	81,89%
	14	36/37	18/18	36/56	3/3	10/12	103/126	81,75%
Random forest	6	38/38	19/19	41/60	4/4	13/13	115/134	85,82%
	8	37/37	18/18	39/59	4/4	13/13	111/131	84,73%
	10	37/37	18/18	38/57	3/3	13/13	109/128	85,16%
	12	37/37	18/18	38/57	3/3	12/12	108/127	85,04%
	14	37/37	18/18	38/56	3/3	12/12	108/126	85,71%

However, for the AMD x86-64 architecture, in most cases it correctly predicts nearly 66%. This makes sense, since it is the CPU architecture that is most widely used on desktop computers and in many cases the malware's source code has been leaked on the Internet, so it may have been modified, mixed with different malware functionalities, etc. This produces new variants of the malware which are created to improve the sample or to try to avoid detection by antivirus systems. In addition, as it is the architecture that is most widely used on personal computers, even a simple port scanner can be classified as malware and fall within the dataset.

## 6. Case study

After grouping similar samples together, we next characterize their functionality and study the most relevant threats as discussed in Section 3.4. For the purpose of this case study, we use a similarity of 80% or higher when building the clusters. We chose this threshold empirically after manually evaluating the resulting clusters. We refer the reader to Section 7 for a discussion on the implications behind our choice.

In this section, we report results that are unknown to the community (i.e., unlabeled samples). Thus, clusters of samples that are not associated with an unlabeled sample are not reported. We then describe a number of Linux-based malware threats targeting the following architectures: AMD, MIPS, ARM, PowerPC and Intel. Table 5 summarizes the findings of the unknown samples for each platform.

### 6.1. AMD x86-64

AMD x86-64 is the architecture with the largest number of unknown threats as well as the most targeted platform, as shown in Table 2. Fig. 5 shows a number of unlabeled clusters of AMD x86-64 Linux-based malware samples (depicted in gray) and their relationship with known threats (a different color per threat). Samples are represented as nodes of the graph and the edges represent that two samples are similar. The samples that have labels are represented by different colors and have some relation with some unlabeled samples. We observe three distinct patterns:

(i) clusters of unlabeled samples alone (clusters #2-#16); (ii) clusters with unknown samples associated with known threats (unnumbered clusters); and (iii) unknown samples that are not similar to any other sample (i.e., singletons). We next present a number of case studies characterizing primarily the most relevant clusters of (i), i.e., unknown clusters alone. However, we also describe cases where a large number of unlabeled samples are clustered together with a small number of labeled samples. While the former are relevant to understanding novel threats, the latter can shed some light on why signatures for “semi-known” are not successfully detecting all other related threats.

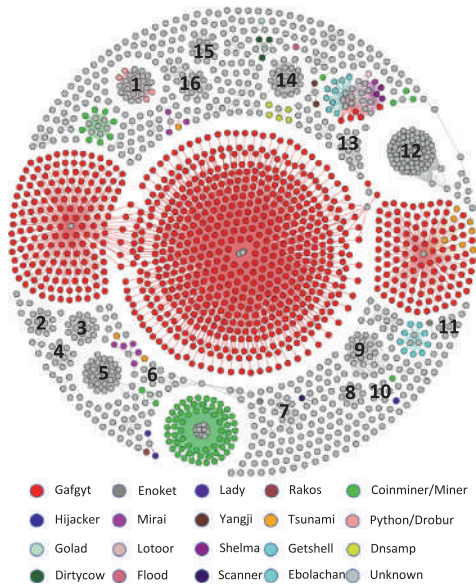
#### 6.1.1. Dropper (clusters #9 and #12)

Our vetting process shows that samples in clusters #9 and #12 in Fig. 5 have a similar behavior, namely they are both droppers. In both clusters, all samples contain a payload that is encrypted and stored in the data section of the executable. We also observe that the decryption routine is similar. In particular, they seem to be using the RC4 algorithm or a variation of it. While these two clusters share a similar backbone, they have been grouped in different clusters mainly because the dynamic features of samples in cluster #9 differ from those in #12. This indicates that samples in these two clusters might belong to the same actor (family and/or botnet), although they participate in different campaigns.

The samples in cluster #12 belong to the largest unlabeled cluster in our dataset. After deciphering the payload of the samples in this cluster, we observe an in-memory bash script that is executed during runtime via the `execvp` syscall. A dump of the script is presented in Listing 2. This routine simply checks whether the target machine has `wget` or `curl` to download a file and prepares for its execution. Also, it informs the server of the user name of the machine, the IP address and operating system. At the time of writing, some samples observed in this campaign are active and the infrastructure still operational. In particular, the executable dropped by the samples of this campaign fetch an AMD-compatible version of a crypto-mining tool. This tool uses CryptoNight Proof-of-Works (PoW) through Stratum, a Pool mining protocol [22]. Pool mining protocols are used to build a distributed partnership to verify the different PoWs in a block.

**Table 5**  
Summary of the results found in the unknown samples for each of the platforms analyzed.

Arch	Type	Comments
AMD x86-64	Dropper	Script Bash encrypted with RC4 and stored in the data section. Once decrypted, the script downloads an executable related to cryptomining campaigns.
	Flooder	Malware used to perform flooding and botnets with flooding capabilities or DDos.
	Shellcode and Exploits	Dirtycow and some type of privilege elevation exploits.
	Goscansh	Malware written in Golang that spreads through SSH services vulnerable to brute force attacks.
ARM 32	Python	Embedded Python interpreter. Some python scripts belong to a Drobur malware, others are benign.
	Botnets	Known malware such as Mirai and Gagyt.
MIPS I	Exploit	Privilege escalation exploits on Android devices.
	Botnets	Known malware such as Mirai, Gagyt, Remaiten, TheMoon, Dofloo and Dnsamp.
PowerPC	Hacktool	Hacking tools such as aircrack or ones to clean the footprints of a system.
	Downloader	Mirai malware downloader.
Intel 80386	Botnets	Known malware such as Mirai, Gagyt, Tsunami and Remaiten. Mainly used to perform DDos attacks.
	Exploit/Shellcode	Generic shellcodes and exploits.
Intel 80386	Virus	ELF File infector known as Vit.
	Backdoor	Ganiw/Setag family, which executes the commands received by the cybercriminals.
	Rex	Malware family written in Golang with mining, flooder and brute force capabilities.



**Fig. 5.** Unknown AMD x86-64 clusters and their relationship with known threats. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

In this way, the complexity required to check a block is distributed among the participants. Therefore, the set of samples in #12 are droppers that are responsible for downloading and executing custom crypto-mining malware. The use of CryptoNight PoWs indicates that droppers are probably part of a larger botnet mining ASIC-resistant cryptocurrencies [23]. ASIC-resistant mining prevents the use of dedicated mining hardware. Thus, using

IoT malware to mine can be profitable, especially when mining through a Pool. Finally, we also observe a few unlabeled samples that are related to known mining campaigns (c.f., green cluster in Fig. 5 for a Coinminer campaign).

#### Listing 2: bash version

```
#!/bin/bash
if [ ! -f "/var/tmp/. "x" ] || [ ! -f "/var/tmp/. "xh" ]; then
#echo File not present. Installing.
if [ -x /usr/bin/wget ]; then
wget --dns-timeout 10 --user-agent="wget" --connect-timeout 20
--read-timeout 30 -q -O $mydir/xx.tgz
--header "Host:www.btcavetheworld.org"
"hxsp://104.24.123.53/xlatest.tgz" 2> /dev/null >> /dev/null
else
if [ -x /usr/bin/curl ]; then
curl -f -A "curl" --connect-timeout 10 -qs --max-time 500
--header "Host:www.btcavetheworld.org"
"hxsp://104.24.123.53/xlatest.tgz"
-o $mydir/xx.tgz >> /dev/null 2> /dev/null
fi
fi
if [ -f $mydir/xx.tgz ]; then
mypwd=$(pwd) 2>/dev/null >> /dev/null
cd $mydir 2>/dev/null >> /dev/null
tar -zxvf xx.tgz 2>/dev/null >> /dev/null
rm -rf xx.tgz 2>/dev/null >> /dev/null
mkdir -p "/var/tmp/. " 2>/dev/null >> /dev/null
mv -f init "$myfile" 2>/dev/null >> /dev/null
mv -f xh "$myfile" 2>/dev/null >> /dev/null
chmod +x "$myfile" "$myxh" 2>/dev/null >> /dev/null
cd "$mypwd" 2>/dev/null >> /dev/null
#echo File is installed
fi
fi
```

When looking at samples in #9, we observe that the main differences lie in the dynamic characteristics, presenting different types of syscalls (e.g., ioctls). This relates to the differences in the behavior of the infected file. Understanding in detail the characterizing features of the executable files dropped by this campaign is the aim of our future work.

#### 6.1.2. Flooder (#2, #5, #7, #10, #11, #14, and #15)

The next most relevant threat is structured around clusters #2, #5, #7, #10, #11, #14, and #15 (see Fig. 5), and it relates to network flooders. We also found a few unlabeled samples assigned to clusters largely populated with samples of known flooders such as: Gagyt,<sup>4</sup> Mirai, Tsunami or Flood. These are tools

<sup>4</sup> For instance, Gagyt (depicted in red in Fig. 5) is a known IoT botnet that has flooding capabilities.

that are typically used to perform Distributed Denial of Service (DDoS) attacks. We have vetted most of the samples in these seven unlabeled clusters. We have observed that all these flooders can be characterized into the following groups according to the protocol they rely on: UDP, SSDP, NTP and Netbios. Similarly to what we observe with the droppers, some of these variants share a common backbone. However, in this case the fact that some clusters share a common backbone does not mean that the clusters are part of different campaigns from the same actor. Instead, they have proliferated after the source code of some of these samples became available [24]. During our analysis, we have been able to track some characterizing features (e.g., strings that are referenced in the disassembly) of the samples in these clusters to IoT flooders with source code that it readily available online. This might explain why this type of threat has proliferated so quickly over the last few years [25,26].

## 6.1.3. Shellcode and Exploits (#4, #6, #8, #13, and #16)

We have observed a wide range of clusters with malware that runs shellcode and executes exploits. The most notable cluster is #4, which uses an exploit that leverages the so-called *dirtycow* vulnerability [27]. This exploit takes advantage of a race condition in the kernel's memory management system to escalate privileges. We performed a reverse search between the exploit found in the samples of cluster #4 and exploit-db [28], a popular open-source repository for exploits. We found that the samples analyzed share code with one exploit-db entry.

Likewise, samples in clusters #6, #8 and #13 belong to some type of privilege elevation exploit. The flow of the logic and the disassembled samples are very similar, but we did not find any payload that would allow us to characterize these clusters. There are similar functions (such as “kernel\_code” or “get\_kernel\_sym”) in all the samples but they are not referenced in the programs that we have vetted. We have not been able to identify any corresponding CVE associated with the exploit either. These clusters probably correspond to more sophisticated malware samples than those observed before.

Finally, cluster #16 is largely related to malware using shellcode. Contrary to what we observed in the previous clusters, we find that most of the samples in this cluster are simple programs. These samples are mainly used to test and debug the shellcode, where they print the size of the binary and make a call at the beginning of the code. Typically, this is used to verify that the code is executed correctly before including an exploit in it or distributing it in online forums [29] or underground markets [30,31]. Within these samples, there are several that invoke system functions using a PowerShell command as an argument. PowerShell has been heavily abused by criminals to easily create cross-platform *fileless* infections — files that are fetched dynamically and executed directly in memory [32].

## 6.1.4. GoScanSSH (#3)

Another prominent cluster is #3, which is also shown in Fig. 5. Samples in this family are related to the GoScanSSH family [33]. This family spreads by searching for servers that are vulnerable to brute force attacks over SSH. The malware is written in the Go language and all samples are stripped binaries with layout obfuscation (i.e., no meaningful function names). However, Go maintains a section called “gopclntab” with a manifest of the functions that might keep the function names if not properly obfuscated. Despite the binary being stripped, we were able to map this cluster to the GoScanSSH family by looking at the “gopclntab” section. An analysis of this section of the binary reveals that the samples in cluster #3 attempt to perform ssh activity, that they also generate wordlists (typical of dictionary attacks), and that the malware maintains a blacklist of domains.

Judging by the VirusTotal *first seen* attribute associated with the samples in this cluster and the first known report modeling this threat [33], we are able to tell that some of these samples have remained undetected for more than a year.

## 6.1.5. Python and Drobrur (#1)

This refers to cluster #1 in Fig. 5. Although the unknown samples in this cluster are related to several labeled samples, there is no consensus among the labeled samples. In particular, some AV vendors label some of these samples as Python and some others as Drobrur. When we go deeper into analyzing these samples, we verify that they all have an independent Python executable. Specifically, the executable has the embedded Python interpreter, as well as its dependencies, so that it can be executed without the need of having Python installed on this architecture. Many IoT platforms are hardened, and it is thus common to find malware samples that piggyback all the dependencies needed to set the scene.

Note that what characterizes the behavior of the samples is the Python script that is executed in each infection, rather than the piggybacked tools. This means that although the distance between these samples might be small, the common factor is generally the use of the same auxiliary tool. The fact that they all have the Python interpreter might mean that these samples belong to the same campaign (e.g., same dropper or the same Pay Per Install botnet). However, it could also be that a given sample is not related to the others at all.

We have randomly chosen several samples and further analyzed them manually. After extracting the Python bytecode and reconstructing the original script, we have verified that some of the samples do not seem to contain harmful code. Some of them contain the same code as the sample labeled as Drobrur, which appears to be an IRC bot. However, two of the samples look benign: one is a dot file editor and the other is a *Taxii Service Connector*. This makes us think that some of the samples detected by some antivirus engines and, therefore, marked as malicious in VirusTotal, use some kind of signature that flags any executable with Python embedded.

## 6.2. MIPS, ARM and PowerPC

We now look at the MIPS, ARM and PowerPC architectures together. Fig. 6 shows the clusters for the different malware samples on the MIPS I architecture. Due to space constraints, we do not visualize clusters for ARM or PowerPC. However, unlabeled clusters are structured in a similar way on ARM and PowerPC to that on MIPS. That is, there are almost no singletons and most of the unlabeled samples are connected to a known family.

For MIPS, as discussed in Section 4, we can observe that Mrblack and Dnsamp are the same sample that has been labeled differently by the antivirus engines. For ARM, we can observe that most of the unlabeled samples are related to Mirai, and on PowerPC with Gafgyt and to a lesser extent with Mirai.

Finally, we have vetted most of the singletons shown on these three architectures. In general, these samples seem to belong to known malware families, corresponding to generic malware samples. However, we have also seen some samples piggybacking exploits that we have not seen before, similarly to what we saw on AMD x86-64 (see Section 6.1.3). We also found some malware piggybacking known hacking tools such as Aircrack.<sup>5</sup> or a tool to clean fingerprints in a system.<sup>6</sup>

<sup>5</sup> Suite of tools to assess WiFi network security widely used to crack WiFi passwords (c.f., <https://www.aircrack-ng.org/>).

<sup>6</sup> UNYUNZ Log Remove & Rename Utility.



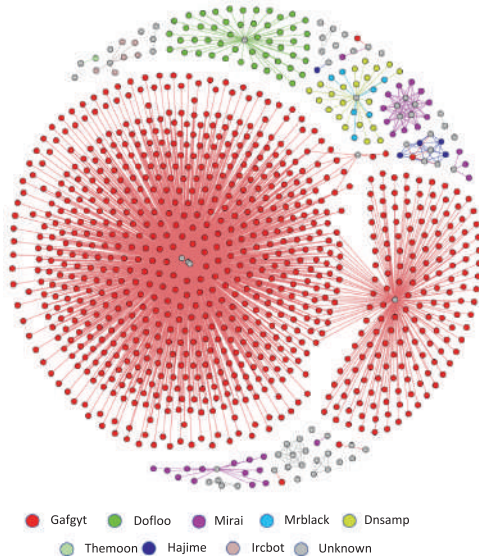


Fig. 6. Clusters on the MIPS I architecture.

### 6.3. Intel 80386

We next focus on the Intel 80386 architecture. Although this case study resembles the one in the previous section (Section 6.2), the main difference here lies in the existence of a relatively large group that contains only unlabeled samples. Fig. 7 shows the relationship between the unlabeled samples and known threats. It can be seen that several unlabeled samples are related mainly to the Setag family and to the Ganiw family. After further analysis, we observe that the Ganiw family is also named after Setag by some AVs. These findings enable us to re-label these new samples and approach the supervised learning phase in Section 4 with a better ground-truth quality.

When looking at the unknown samples alone, we first focus on the larger clusters. The largest cluster belongs to a family called Vit.4096, which is a virus that has been around for over two decades. This virus replicates itself by adding 4096 bytes after the header of other ELFs in the system. The first reference to the virus dates back to 1996 [34]. However, signatures for this malware only appear in VirusTotal for the first time in 2017. In addition to this cluster, there are two other small clusters of 4 samples, each of which also belongs to this family. When reversing all these samples, we can confirm that they do indeed belong to the Vit family. We note that there is some small difference in the number of functions among the samples of different groups. The samples have a different cyclomatic complexity in some of their functions. Furthermore, there is a small difference in the overall number of functions among samples from different clusters. All the samples have a unique Indicator of Compromise (IoC) – the string “Vi324.tmp” appears across all the clusters and it is used as a file name for a temporary file during virus replication.

When looking at other clusters we find: (i) a Bitcoin Wallet Bot; (ii) the Rex family [35], which is malware with mining capabilities that propagates by exploiting wordpress, drupal, and magento vulnerabilities; (iii) flooders; and (v) more brute force attacks on SSH services as well as on telnet services.

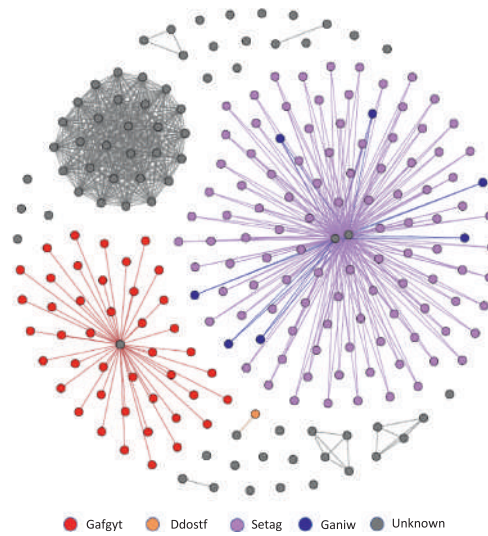


Fig. 7. Clusters on the Intel 80386 architecture.

Finally, when looking at the singletons we observe malware carrying several exploits and miscellaneous shellcode.

## 7. Discussion

Our work is based on the most representative dataset of Linux-based malware collected by the community to date [2]. In this section, we first discuss the main limitations of our work. Despite these limitations, our findings provide a fresh understanding of recent trends in Linux-based malware. In particular, we focus on characterizing a large number of samples that were unknown to the community. The set of non-characterized threats constitute 15% of the samples in [2]. The analysis performed in the discovery phase can help in the modeling of TTPs (Tactics, Techniques and Procedures), which is key in the attribution of malware. Matching different malware samples may identify different actors and their TTPs [36], which is very helpful in supporting incident response teams [37]. To the best of our knowledge, there is nothing published on malware attribution issues in IoT environments. Thus, we finally report key findings derived from vetting the most prevalent threats in the non-characterized set.

### 7.1. Threats to validity

The proliferation of IoT malware is currently bound by the architecture of the different types of IoT devices. Based on this, we first describe how this affects the scope of our analysis and we then discuss the limitations of our system.

**Scope.** IoT malware targets a wide range of devices [38], each having a particular architecture. Well-known processor architectures are MIPS and ARM, but also AMD and Intel have recently been competing to provide energy-efficient chips [39,40]. On top of the hardware, these devices are increasingly incorporating additional layers of software to deliver cross-platform controllers and/or operating systems. This is the case, for instance, of Arduino

Yun,<sup>7</sup> which runs Linux on top of an Arduino Leonardo. Another example is Google Brillo,<sup>8</sup> which has a scrubbed-down version of Android (also Linux-based). Currently, a wide range of the devices susceptible to being attacked by IoT malware run a flavor of Linux. Thus, architectures commonly used in desktop computers can also be targeted by the same malware and are part of this ecosystem. Likewise, malware developed for a desktop computer, could easily be re-compiled to run, for example, on top of a Linux-based ARM device. In our work, we approximate the analysis of the IoT malware landscape by focusing mainly on Linux-based malware. In particular, we look at this threat through the lens of a dataset collected by Cozzi et al. [2] in 2018. Although this dataset has been collected to evaluate a malware detection sandbox tailored to different IoT architectures, we note that it has been collected using protected malware Threat Intelligence. Thus, we acknowledge that there might be some multi-purpose malware samples that could also run on desktop computers. However, we also note that when the malware is collected through honeypots, this figure will be very small or even non-existent.

**Limitations.** IoT malware is an emerging threat and, therefore, does not yet have features as advanced as its predecessors on platforms such as Windows or Android. This leads to the use of both static characteristics and the use of disassembler n-gram sequences to perform correctly. In the near future, we expect to see an increase in the sophistication of IoT malware with the adoption of obfuscation or packing, thus hindering static analysis and limiting the visibility of our system. Another weakness derived from the use of static features is caused by binaries that are not necessarily malicious and that are embedded in the malware, such as those that are built with PyInstaller [41]. Tackling this challenge might require dedicated techniques similar to those proposed for dealing with repackaged malware in Android [42]. In our work, we partly overcome these limitations in two ways: (i) we first use dynamic analysis to be able to characterize malware intended to hinder static analysis; and (ii) we maintain a high similarity threshold to avoid clustering samples with different payloads but which piggyback the same library.

## 7.2. Key-findings

Our system builds a guided system to better characterize unknown threats in IoT malware. We next describe the main trends, focusing on those obtained from previously unknown threats.

**Trends.** We found that crypto-mining malware is targeting IoT platforms. This provides cybercriminals with an infrastructure connected 24 h a day that allows them to mine cryptocurrencies in a distributed manner and obtain profits without investing in hardware or paying the electricity bill. Originally, the malware aimed at IoT platforms had the main purpose of creating a network of bots to perform denial of service attacks or to make money by selling access to a network of bots on the black market (Botnet-as-a-service) [43]. Nowadays, this is changing and cybercriminals are taking advantage of the resources of the infected devices. These stolen resources are currently used to mine cryptocurrencies and, therefore, obtain a greater economic benefit [29]. The study of the trends in malware targeting IoT devices could help develop forensic tools and methodologies tailored to such threats. This is because the characterization of IoT malware can lead to the discovery of evidence that can change the course of an investigation (i.e., increasing the percentage of CPU usage or battery consumption as evidence of infection by ransomware or crypto-mining malware [44]). Thus, the analysis

performed can put forensic analysts in an advantaged position in the prosecution of criminals using the Trojan Horse Defense.

**Sophistication.** We have observed that the level of sophistication of IoT malware varies considerably. While there is certainly a large number of families with little sophistication, we are increasingly seeing more complex malware families (e.g., droppers and exploits, as reported in Sections 6.1.1 and 6.1.3, respectively). This is supported by recent commercial reports, such as the one in [45], which provide anecdotal evidence of this phenomenon.

**Infrastructure.** We also found that samples in a wide range of clusters rely on publicly available repositories such as exploit-db and GitHub (flooderscode) to manufacture new variants of malware. For example, the code of some of the most widespread botnets is available on the Internet [46], such as Mirai, Bashlite (Gafgyt), or pnsan. This encourages the development of novel samples that simply reuse part of the available code. This leads to the rapid proliferation of new variants with little investment and high economic returns.

## 8. Related work

Celeda et al. [47] present an analysis of the Chuck Norris malware, which they discovered due to the growth of port 23 scanners in their networks. The authors prepared a vulnerable device for the botnet and monitored all the network connections that were generated in that device for the purpose of an exhaustive analysis of the botnet.

Bohio [48] performs a technical analysis of the Dofloo malware, using static and dynamic code analysis. It details the commands supported by the malware as well as the communication mechanisms with the C&C. It also proposes a network traffic detection signature for Snort as well as commitment indicators to detect malware in a compromised system.

Wang et al. [49] distinguish two main infection methods for malware that is focused on IoT platforms. The authors use Mirai as an example of malware that uses brute force to infect devices, and Bashlite and Darloz as examples to analyze malware that exploits vulnerabilities for its propagation.

In [50] the authors carry out a review of the state of the art of botnets, analyzing the techniques used by the Mirai and Hajime botnets. Finally, they propose countermeasures for the detection of these botnets.

In [25] a complete study on the Mirai botnet is presented in which the authors analyze its evolution, before and after the release of its source code. They describe the main devices affected and the different types of attacks that the botnet carries out, as well as the targets which it is aimed at. Finally, as a case study, they analyze attacks on three Mirai victims.

Edwards et al. [51] perform a complete analysis of the Hajime botnet, detailing everything from the recognition and infection phase of new victims to the format of custom files used to store the configuration and payload. In addition, they describe the types of messages that Hajime supports. Finally, they provide countermeasures for detecting the botnet.

Herwig et al. [52] perform a study of the Hajime botnet from different points of view, such as the geographical locations of the infected devices and the types of devices. In their study, they also analyze the size of the botnet and the churn rates, as well as the TR-064 vulnerability that Hajime exploits by analyzing the queries collected from a DNS root server.

De Donno et al. [53] perform a study on the state of the art of malware in the IoT, presenting a classification of DDoS attacks according to different features such as the botnet architecture, protocol, scanning strategy, etc. They also present a description of the main families of botnets in the IoT and the relationships between them, showing the types of DDoS attacks that each one

<sup>7</sup> <https://www.arduino.cc/en/Guide/ArduinoYun>

<sup>8</sup> Code-named Android Things <https://developer.android.com/things>.

is capable of performing, the CPU architectures that support them and the DDoS architecture of the botnet. Finally, they present an analysis of the Mirai botnet.

In [2], the authors present the details of a platform they have developed that is focused on malware analysis based on Linux. Their platform supports the main target architectures of current IoT malware. In addition, they describe in detail different techniques used by the malware in Linux as well as the statistics of how many of the samples out of the total that comprised their study implement these techniques.

In [54], a study on malware in the IoT and its families is presented, summarizing the size of botnets as well as the estimated time they remain active. It also includes an analysis of the rules for IDS and the time window from when the malware appeared until the first rules appeared. In addition, they present a series of errors and inconsistencies found in their studies as well as the ambiguity of vulnerability references, analysis information, etc. Finally, they present a framework for analyzing malware dynamically that is based on the open source tool Cuckoo Box [55].

Cosa Nostra is a toolkit for clustering malware created by Joxean Koret [56]. The tool generates a call graph signature for each sample of malware analyzed using the complex cyclomatic of all the functions and assigning a prime number to that cyclomatic complexity. Finally, it generates a hash based on the multiplication of those prime numbers. When two hashes are equal it is considered that the samples are structurally equal, and if they are different, it breaks down the hash into its prime factors to determine how much they differ from each other. It allows the creation of phylogenetic trees of malware samples that are structurally similar. This can be visualized in a Web GUI.

Isawa et al. [57] propose the use of static features to compute the similarity between samples of IoT malware, since the extraction of static features consumes less time than obtaining characteristics based on their behavior. For their experiments, they build similarity matrices based on the use of *n*-grams for both disassembled code and system call traces. Finally, they visualize these matrices, verifying that the use of disassembled code works well for the classification of malware in the IoT.

Nguyen et al. [58] compare three approaches for malware detection using Convolutional Neural Networks (CNN). The first is based on fixed-size byte sequences, the second uses fixed-size color images, and the third assembler instruction sequences of variable size. Their tests are performed on 1000 samples of malware and 1000 goodware samples for the x86 architecture, showing by their experimental results that the approaches based on CNN work well in the detection of malware in the IoT.

## 9. Conclusions

In this paper we have presented a study of malware that targets IoT platforms. Through data analysis, we extracted static and dynamic features to systematically characterize malware into different threats. The proposed methodology allows the identification of new malware samples and the relationships they maintain with previous ones. Our evaluation over a dataset of labeled samples shows that our system can accurately perform this task.

Our methodology allows the extraction of knowledge about large groups of connected samples by analyzing some of their samples and extrapolating the results obtained. We have used this to investigate a number of unlabeled malware samples found in the wild. Where applicable, we have associated unknown clusters with known threats. This showed that the current detection mechanisms deployed by commercial AntiVirus systems are behind in the arms race. We have also gone one step further and

studied each of the unknown clusters by using state-of-the-art reverse engineering techniques and our expertise as malware analysts. In this way, we were able to verify that the relationship formed by samples from the same group was correct, identifying the groups with the highest number of samples. We have also provided an in-depth analysis of what the most recent unknown trends are. We have shown, for instance, that crypto-mining malware is currently attacking the IoT infrastructure. Our hope is that this characterization will help the community to devise better detection strategies against these specific threats. Finally, in order to foster the development of these strategies, we have released the characterization we have produced for each of the clusters.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## CRedit authorship contribution statement

**J. Carrillo-Mondéjar:** Conceptualization, Software, Validation, Investigation, Writing - original draft, Visualization. **J.L. Martínez:** Conceptualization, Methodology, Investigation, Writing - original draft, Writing - review & editing, Supervision, Project administration, Funding acquisition. **G. Suarez-Tangil:** Conceptualization, Methodology, Investigation, Resources, Writing - original draft, Writing - review & editing, Supervision.

## Acknowledgments

This work has been supported by the MINECO, Spain and European Commission (FEDER funds) under project RTI2018-098156-B-C52, by the JCCM, Spain under the project SB-PLY/17/180501/000353 and by the Spanish Education, Culture and Sports Ministry under grant FPU 17/03105.

## References

- [1] Gartner, Gartner says 6.4 billion connected, 2015, <http://www.Gartner.Com/Newsroom/Id/3165317>. (Retrieved February 2019).
- [2] E. Cozzi, M. Graziano, Y. Fratantonio, D. Balzarotti, Understanding linux malware, in: 2018 IEEE Symposium on Security and Privacy, SP, 2018, pp. 161–175, <http://dx.doi.org/10.1109/SP.2018.00054>.
- [3] M. Kuzin, V.K. Yaroslav Shmelev, New trends in the world of IoT threats, 2018, Available online.
- [4] F. Marturana, S. Tacconi, A Machine Learning-based Triage methodology for automated categorization of digital media, Digit. Invest. 10 (2) (2013) 193–204.
- [5] D. Quick, K.-K.R. Choo, Impacts of increasing volume of digital forensic data: A survey and future research challenges, Digit. Invest. 11 (4) (2014) 273–294.
- [6] A. Holzinger, P. Kieseberg, E. Weippl, A.M. Tjoa, Current advances, trends and challenges of machine learning and knowledge extraction: From machine learning to explainable AI, in: A. Holzinger, P. Kieseberg, A.M. Tjoa, E. Weippl (Eds.), Machine Learning and Knowledge Extraction, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2018, pp. 1–8, [http://dx.doi.org/10.1007/978-3-319-99740-7\\_1](http://dx.doi.org/10.1007/978-3-319-99740-7_1).
- [7] S. Bowles, J. Hernandez-Castro, The first 10 years of the Trojan Horse defence, Comput. Fraud Secur. 2015 (1) (2015) 5–13.
- [8] M. Sebastián, R. Rivera, P. Kotzias, J. Caballero, Avclass: A tool for massive malware labeling, in: F. Monrose, M. Dacier, G. Blanc, J. Garcia-Alfaro (Eds.), Research in Attacks, Intrusions, and Defenses, Springer International Publishing, Cham, 2016, pp. 230–253.
- [9] Virus Total, 2019, [Online] <https://www.virustotal.com>. (Accessed February 2019).



- [10] M. Hurier, K. Allix, T.F. Bissyandé, J. Klein, Y. Le Traon, On the lack of consensus in anti-virus decisions: Metrics and insights on building ground truths of android malware, in: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2016, pp. 142–162.
- [11] A.H. Watson, D.R. Wallace, T.J. McCabe, Structured testing: A testing methodology using the cyclomatic complexity metric, 500, (235) US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1996.
- [12] J. Leskovec, A. Rajaraman, J.D. Ullman, Mining of Massive Datasets, Cambridge University Press, 2014.
- [13] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
- [14] I.H. Witten, E. Frank, M.A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann Publishers Inc., Burlington, MA, 2011.
- [15] J. Laaksonen, E. Oja, Classification with learning k-nearest neighbors, in: Proceedings of International Conference on Neural Networks, Vol. 3, ICNN'96, 1996, pp. 1480–1483, <http://dx.doi.org/10.1109/ICNN.1996.549118>.
- [16] J.R. Quinlan, C4.5: Programs for Machine Learning, Elsevier, 2014, Google-Books-ID: b3ujBQAAQBAJ.
- [17] M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt, B. Scholkopf, Support vector machines, IEEE Intell. Syst. Appl. 13 (4) (1998) 18–28, <http://dx.doi.org/10.1109/5254.708428>.
- [18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
- [19] M. Stone, Cross-validatory choice and assessment of statistical predictions, J. R. Stat. Soc. Ser. B Stat. Methodol. 36 (2) (1974) 111–147, URL <http://www.jstor.org/stable/2984809>.
- [20] Y.-W. Chen, C.-J. Lin, Combining SVMs with various feature selection strategies, in: Feature Extraction, Springer, 2006, pp. 315–324.
- [21] L.v.d. Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (Nov) (2008) 2579–2605, URL <http://www.jmlr.org/papers/v9/vandermaaten08a.html>.
- [22] R. Recabarren, B. Carbunar, Hardening stratum, the bitcoin pool mining protocol, Proc. Priv. Enhanc. Technol. 2017 (3) (2017) 57–74.
- [23] S. Pastrana, G. Suarez-Tangil, A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth, 2019, arXiv preprint arXiv:1901.00846.
- [24] ring04h, ring04h/WebTerror, URL <https://github.com/ring04h/WebTerror>.
- [25] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J.A. Halderman, L. Invernizzi, M. Kallitsis, et al., Understanding the mirai botnet, in: 26th (USENIX) Security Symposium (USENIX) Security 17, 2017, pp. 1093–1110.
- [26] C. Kolias, G. Kambourakis, A. Stavrou, J. Voas, DDoS in the IoT: Mirai and other botnets, Computer 50 (7) (2017) 80–84.
- [27] CVE Details, CVE-2016-5195, URL <https://www.cvedetails.com/cve/CVE-2016-5195/>.
- [28] FireFart, Linux kernel 2.6.22 < 3.9 - 'Dirty COW', 2016, Exploit Database, URL <https://www.exploit-db.com/exploits/40839>.
- [29] S. Pastrana, D.R. Thomas, A. Hutchings, R. Clayton, Crimebb: Enabling cybercrime research on underground forums at scale, in: Proceedings of the 2018 World Wide Web Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2018, pp. 1845–1854.
- [30] L. Allodi, Economic factors of vulnerability trade and exploitation, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2017, pp. 1483–1499.
- [31] R. Van Wegberg, S. Tajalizadehkhoob, K. Soska, U. Akyazi, C.H. Ganan, B. Klievink, N. Christin, M. Van Eeten, Plug and prey? measuring the commoditization of cybercrime via online anonymous markets, in: 27th (USENIX) Security Symposium (USENIX) Security 18, 2018, pp. 1009–1026.
- [32] D. Hendler, S. Kels, A. Rubin, Detecting malicious PowerShell commands using deep neural networks, in: Proceedings of the 2018 Asia Conference on Computer and Communications Security, ACM, 2018, pp. 187–197.
- [33] E. Brumaghin, A. Williams, A. Zidoumba, Forgot about default accounts? No worries, GoScanSSH didn't, 2018, URL <https://blog.talosintelligence.com/2018/03/goscanssh-analysis.html>.
- [34] Hispasec, Virus para Linux, URL <https://unaaldia.hispasec.com/1999/02/virus-para-linux.html>.
- [35] B. Ancel, From website-locker to DDoS: Rex1, 2016, URL <https://thisissecurity.stormshield.com/2016/08/17/from-website-locker-to-ddos-rex/>.
- [36] J.S. Davis II, B. Boudreaux, J.W. Welburn, J. Aguirre, C. Ogletree, G. McGovern, M.S. Chase, Stateless Attribution, RAND Corporation, 2017.
- [37] W. Tounsi, H. Rais, A survey on technical threat intelligence in the age of sophisticated cyber attacks, Comput. Secur. 72 (2018) 212–233.
- [38] M. Ammar, G. Russello, B. Crispo, Internet of things: A survey on the security of iot frameworks, J. Inf. Secur. Appl. 38 (2018) 8–27.
- [39] E. Blem, J. Menon, K. Sankaralingam, Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures, in: 2013 IEEE 19th International Symposium on High Performance Computer Architecture, HPCA, IEEE, 2013, pp. 1–12.
- [40] M.R. Nakhkash, T.N. Gia, I. Azimi, A. Anzanpour, A.M. Rahmani, P. Liljeberg, Analysis of Performance and Energy Consumption of Wearable Devices and Mobile Gateways in IoT Applications, COINS, 2019.
- [41] PyInstaller Development Team, PyInstaller Quickstart – PyInstaller bundles python applications, URL <https://www.pyinstaller.org/>.
- [42] G. Suarez-Tangil, G. Stringhini, Eight years of rider measurement in the android malware ecosystem: evolution and lessons learned, 2018, arXiv preprint arXiv:1801.08115.
- [43] T. Seals, TheMoon rises again, with a botnet-as-a-service threat, URL <https://threatpost.com/themoon-botnet-as-a-service/141393/>.
- [44] A. Azmoodeh, A. Dehghantanha, M. Conti, K.-K.R. Choo, Detecting crypto-ransomware in IoT networks based on energy consumption footprint, J. Ambient Intell. Humaniz. Comput. 9 (4) (2018) 1141–1152, <http://dx.doi.org/10.1007/s12652-017-0558-5>.
- [45] J. Kroustek, V. Ilushin, A. Shirokova, J. Neduchal, M. Hron, Torii botnet – Not another Mirai variant, URL <https://blog.avast.com/new-torii-botnet-threat-research>.
- [46] F. Ding, Iot-malware, URL <https://github.com/iffding/iot-malware>.
- [47] P. Celeda, R. Krejci, J. Vykopal, M. Drasar, Embedded malware-an analysis of the chuck norris botnet, in: 2010 European Conference on Computer Network Defense, IEEE, 2010, pp. 3–10.
- [48] M. Bohio, Analyzing a Backdoor/Bot for the MIPS Platform, Tech. Rep., SANS Institute, 2015.
- [49] A. Wang, R. Liang, X. Liu, Y. Zhang, K. Chen, J. Li, An inside look at IoT malware, in: International Conference on Industrial IoT Technologies and Applications, Springer, 2017, pp. 176–186.
- [50] G. Kambourakis, C. Kolias, A. Stavrou, The Mirai botnet and the IoT Zombie Armies, in: MILCOM 2017 - 2017 IEEE Military Communications Conference, MILCOM, 2017, pp. 267–272, <http://dx.doi.org/10.1109/MILCOM.2017.8170867>.
- [51] S. Edwards, i. Profetis, Hajime: Analysis of a decentralized internet worm for IoT devices, Technical Report, 2016, URL <https://security.rapid7networks.com/publications/2016-10-16/hajime.pdf>.
- [52] S. Herwig, K. Harvey, G. Hughey, R. Roberts, D. Levin, Measurement and analysis of Hajime, a peer-to-peer IoT botnet, in: Network and Distributed System Security (NDSS) Symposium, 2019, <http://dx.doi.org/10.14722/ndss.2019.23488>, URL [https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019\\_02B-3\\_Herwig\\_paper.pdf](https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_02B-3_Herwig_paper.pdf).
- [53] M. De Donno, N. Dragoni, A. Giarretta, A. Spognardi, DDoS-capable IoT malwares: Comparative analysis and mirai investigation, Secur. Commun. Netw. 2018 (2018).
- [54] A. Costin, J. Zaddach, IoT Malware: Comprehensive Survey, Analysis Framework and Case Studies, BlackHat USA, 2018.
- [55] C. Guarnieri, Cuckoo Sandbox - Automated Malware Analysis, URL <https://cuckoosandbox.org/>.
- [56] J. Koret, Cosa nostra, URL <https://github.com/joexankoret/cosa-nostra>.
- [57] R. Isawa, T. Ban, Y. Tie, K. Yoshioka, D. Inoue, Evaluating disassembly-code based similarity between IoT malware samples, in: 2018 13th Asia Joint Conference on Information Security, AsiaCIS, 2018, pp. 89–94, <http://dx.doi.org/10.1109/AsiaCIS.2018.00023>.
- [58] K.D.T. Nguyen, T.M. Tuan, S.H. Le, A.P. Viet, M. Ogawa, N. Le Minh, Comparison of three deep learning-based approaches for IoT malware detection, in: 2018 10th International Conference on Knowledge and Systems Engineering, KSE, IEEE, 2018, pp. 382–388.



**Javier Carrillo Mondéjar** received a BSc degree in Computer Science and a Master's degree in Advanced Computer Science from the University of Castilla-La Mancha, Spain, in 2016 and 2017, respectively. Currently, he is enrolled full-time on the PhD Program in Advanced Information Technology at this university. In 2016, he joined the Computer Architecture and Technology Group of the Informatics Research Institute of Albacete (ISA) as a researcher. His research interests are related to malware detection and classification techniques, as well as the methods used in malware to spread and remain hidden in computer systems. He has also been a visiting researcher at King's College London.



**José Luis Martínez** received his M.Sc. and Ph.D. degrees in Computer Science and Engineering from the University of Castilla-La Mancha (Spain) in 2007 and 2009, respectively. In 2005, he joined the Department of Computer Engineering at the University of Castilla-La Mancha, where he was a researcher in the Computer Architecture and Technology group at the Albacete Research Institute of Informatics (I3A). In 2010, he joined the department of Computer Architecture at the Complutense University in Madrid, where he was an assistant lecturer. In 2011, he rejoined the Department

of Computer Engineering of the University of Castilla-La Mancha, where he is currently a lecturer and he is accredited as full professor. His research interests include video coding and transcoding, and topics related to security. He has also been a visiting researcher at the Florida Atlantic University, Boca Raton (USA)

and the Centre for Communication System Research (CCSR), at the University of Surrey, Guildford (UK). He has over 100 publications in these areas in international refereed journals and conference proceedings.



**Guillermo Suarez-Tangil** is a lecturer at King's College London (KCL). His research focuses on systems security and malware analysis and detection. In particular, his area of expertise lies in the study of smart malware, ranging from the detection of advanced obfuscated malware to automated analysis of targeted malware. Before joining KCL, he was senior research associate at University College London (UCL), where he was also actively involved in other research areas involved with detecting and preventing Mass-Marketing Fraud.

## CHAPTER 7

# HALE-IoT: HArdening LEgacy Internet-of-Things devices via retrofitting defensive firmware modifications and implants

- **Title:** HALE-IoT: HArdening LEgacy Internet-of-Things devices via retrofitting defensive firmware modifications and implants.
- **Authors:** J. Carrillo-Mondéjar, Hannu Turtiainen, Andrei Costin, J.L. Martínez, and G. Suarez-Tangil.
- **Type:** Journal paper.
- **Journal:** Internet of Things Journal.
- **Publisher:** IEEE.
- **ISSN:** 2327-4662.
- **Status:** Minor revision.
- **JCR IF/ranking:** 10.238/Q1 (JCR2021).

# HALE-IoT: HARDening LEGacy Intenet-of-Things devices via retrofitting defensive firmware modifications and implants

J. Carrillo-Mondéjar, Hannu Turtiainen, Andrei Costin, J.L. Martínez, and G. Suarez-Tangil

**Abstract**—IoT devices and their firmware are notoriously known for their lifelong vulnerabilities. As we see a sharp increase in the number of infected devices, we also see how vendors fail at releasing patches at a competitive pace. Despite security in IoT being an active area of research, prior work mainly focuses on vulnerability detection and exploitation, threat modelling, and protocol security. However, these methods are ineffective in preventing attacks against legacy and End-Of-Life devices that are already vulnerable. Hardening emerges as an effective solution to provide IoT devices with an additional layer of defense. Unfortunately, existing approaches are limited in their scope and their lack of systematic techniques. Systematically hardening vulnerable IoT devices and their firmware is important due to the diversity in the architectures of the devices and the heterogeneity of the firmware images, but for the same reasons faces many challenges. Moreover, IoT firmware modifications research is under-represented and mainly focuses on implementing and demonstrating the potential of malicious modifications, rather than addressing defensive and protective mechanisms for firmware hardening.

In this paper, we bridge these gaps through the design of *HALE-IoT* — a generically-applicable systematic approach to HARDening LEGacy IoT non-low-end devices via retrofitting defensive firmware modifications *without access to original source code*. *HALE-IoT* approaches this non-trivial task via binary firmware reversing and modification and is underpinned by a semi-automated toolset that aims at keeping cybersecurity hale, focusing both on modern and, especially, on legacy or obsolete IoT devices as they become increasingly prevalent. To evaluate the effectiveness and efficiency of *HALE-IoT*, we apply it to a wide-range of IoT devices. For this, we retrofit the firmware of 395 images with defensive implants containing an intrusion prevention system in the form of a Web Application Firewall (for prevention of web-attack vectors), and a HTTPS-proxy (for latest and full end-to-end HTTPS support) using emulation. We also test our approach in four physical devices, where we show that *HALE-IoT* successfully runs on protected and quite constrained devices featuring as low as 32MB of RAM and 8MB of storage. Overall, in our evaluation, we achieve a good performance and reliability with a remarkably accurate detection and prevention rate for attacks coming from both real CVEs and synthetic exploits.

**Index Terms**—cybersecurity, IoT, devices, legacy, end-of-life, EOL, firmware, firmware modification, retrofit security, defensive techniques, WAF, HTTPS, SSL-proxy

J. Carrillo-Mondejar and J.L. Martínez are with University of Castilla-La Mancha, Albacete, Spain, e-mail: {javier.carrillo,joseLuis.Martinez}@uclm.es

Hannu Turtiainen and Andrei Costin are with University of Jyväskylä, Jyväskylä, Finland e-mail: {turthzu,ancostin}@jyu.fi

G. Suarez-Tangil is with IMDEA Networks Institute, Madrid, Spain e-mail: guillermo.suarez-tangil@imdea.org

## I. INTRODUCTION

Internet of Things (IoT) devices have notoriously vulnerable firmware [1]–[3]. Exploiting these vulnerabilities is oftentimes trivial, an example being the case of the infamous Mirai botnet [4]. Unfortunately, keeping the firmware of these devices updated is challenging. First, in many cases a firmware update or a patch is simply not available. This is a prevalent problem due to the number of legacy devices connected to the internet [4]–[9]. Second, firmware is in-built to the devices, while automated — Over-The-Air (OTA) — firmware updates are generally not implemented or still have limited adoption [10]. Updating IoT devices, when and if available, may require a fairly technical manual intervention, including having admin access and re-flashing the device, and can prove challenging and error prone even for experienced users. These difficulties foster a culture of bad security hygiene around IoT. As a result many IoT devices are left vulnerable with long-lasting dire consequences [11]. For instance, researchers recorded over 1.5 billion attacks against IoT devices in the first half of 2021 [12]. In this context, just one single vulnerability (CVE-2021-28372) [13] was responsible of affecting around 83 million devices, while some others (e.g., CVE-2013-7471) have been active for years and are still seen in the wild.<sup>1</sup>

In the absence of regular updates, bastioning IoT devices and hardening potentially vulnerable services emerges to offer a first-line defense. The Center for Internet Security (CIS) offers pre-hardened images and hardening checklists that have been adopted by the research community, which includes mechanisms to disable non-essential services [14]. While this reduces the attack surface, essential services may still suffer vulnerabilities [15]. Furthermore, existing approaches like [14], [16] do not address the constraints and the heterogeneity of modern IoT devices. Thus, applying off-the-land defenses at the network level, like third-party firewalls, has been previously the subject of some research [17]–[19]. The next line of defense include retrofitting active [20], [21], and retroactive defenses [4], [22]. Retrofitting defenses into an IoT devices offers the same advantages as general-purpose hardening, while allowing the enhancement of its security mechanisms even without the support of the manufacturer.

Retrofitting security to legacy IoT devices faces many challenges. First, the firmware stock is large and heterogeneous, and non-generic solutions hinder the adoption of this defense. Second, injecting externally compiled code then expecting it

<sup>1</sup>See timeline in: <https://vuldb.com/?id.136365>

to tightly co-exist with the firmware is a challenging and error prone process. Third, IoT devices generally have constrained resources and I/O interfaces, hence cannot easily accommodate arbitrary defensive solutions (e.g., IDS, antivirus) that are useful at defending traditional computing devices (e.g., PCs, laptops, servers). Constrained by such challenges, existing approaches are limited in the scope of their implementation. For instance, Cui and Stolfo [20] present a binary-patching tool called *Doppelgänger* that only offers in practice protection against rootkits. *Doppelgänger* is a system that computes hashes of memory regions where “critical system processes” are mapped, and is essentially a memory integrity monitor. The system then monitors changes in the hashes as a mechanism to detect function hooking and other types of code injection. While *Doppelgänger* can compute and monitor hashes for any arbitrary memory region, identifying and understanding those regions requires considerable human expertise. Thus, approaches such as *Doppelgänger* [20] do not scale in practice, and can not be systematically deployed. Other works focus on hardening a particular type of web applications against XSS and SQLi [15] attacks. However, these approaches rely on modifying the web interpreters, which requires: i) deep software modifications; ii) intimate knowledge of the targeted technology (e.g., PHP); iii) pre-automation tedious taint annotations (e.g., sensitive sinks); iv) and access to source code.

To address existing shortcomings, we use the notion similar to Symbiotic Embedded Machines [20] to design a systematic approach to harden legacy non-low-end IoT devices. We retrofit complex defense systems into raw firmware binaries via systematic yet minimally-intrusive low-level modifications. Our approach differs from the state-of-the-art in several ways. First and foremost, we design a generically-applicable framework to provide reliable security and protective standards to legacy firmware. Second, we develop a systematic testing methodology that constitute the first benchmark to assess the effectiveness of retrofitting defensive firmware modifications.

We develop a cross-platform system, called *HALE-IoT*, that at the time of writing successfully runs at least on MIPS64, MIPSel, ARMel, and Intel 80386 architectures. *HALE-IoT* incorporates several industry-standard security tools. We make a battery of tests using real-world attacks, particularly focusing its evaluation on fuzzing the web interface for two main reasons. On the one hand, the web-interfaces are well known to be exposed and lacking security in many aspects [23], [24], while IoT devices are often proven to have their web-interfaces highly vulnerable and exposed [2], [3]. On the other hand, several studies reported that compared to other online services, the (I)IoT devices much more often run missing, lax, or insecure SSL/TLS implementations [25]–[28], and see insignificant progress towards secure TLS [25].

We note that while our evaluation reports detection rates, its main focus is not to assess how well *HALE-IoT* detects and prevents real-world attacks. In essence, *HALE-IoT* embeds industry-standard protection mechanism such as the ones that have been widely tested before, e.g., Web Application Firewalls (WAF) like Raptor [29]. Instead, we aim at assessing that the retrofitted defensive mechanism can effectively (e.g., detect and protect) and correctly co-exist within the retrofitted

firmware without preventing the normal use of the system (e.g., not crashing it). This is important as software projects such as full-fledged WAF are fairly sophisticated. To the best of our knowledge, no prior work has attempted and assessed the feasibility of implanting sophisticated frameworks<sup>2</sup> into IoT firmware.

We evaluated the effectiveness of our methodology using 395 different firmware images from a wide-range of vendors, including D-Link, Netgear, Linksys, TRENDnet, and OpenWrt. We emulate those 395 firmwares using a similar procedure as in the state-of-art works [2], [3]. Due to the limitations behind acquiring hardware for all vendors, we restrict our bare-metal evaluation to four physical devices (Table IV) featuring 32MB to 1024MB of RAM and 8MB to 4096MB of storage, while representing both ARM and MIPS architectures as well as open-source and proprietary hardware and firmware. At the same time, we note that our current efforts do not attempt test *HALE-IoT* in the “long trail” of architectures (e.g., niche architectures and targets not supported by GCC,<sup>3</sup> nor on architectures that are supported but are not widespread, the same way we do not claim that *HALE-IoT* works on low-end and very constrained targets such as MSP430). A more extensive evaluation is part of future work.

Our main contributions are summarized as follow:

- 1) We develop a generic methodology supported by a *system architecture and a reference implementation* for hardening legacy IoT devices via defensive firmware retrofitting and implants. To the best of our knowledge, it is a first of its kind.
- 2) We evaluate the effectiveness of our methodology by *testing it on potentially vulnerable and insecure web-interfaces* of a large and diverse set of IoT vendors and devices.
- 3) We identify and derive several core challenges of this problem space that require further attention and research.

The rest of the paper is organized as follows. We first introduce and detail the *HALE-IoT* architecture and methodology in Section III. In Section IV we detail the experimental setup, the datasets. We present the testing methodology and the results in Section V. We then discuss challenges and future improvements in Section VII. We present and discuss related work in Section VIII. Finally, we conclude the paper with Section IX.

## II. THREAT OVERVIEW

There are nowadays millions of devices connected to the Internet that shape the way users interact with technology. These devices provide several attractive features that make them popular. Unfortunately, many of these devices lack basic security and privacy protections. This leaves IoT devices exposed to major security issues ranging from insecure configurations and protocols (i.e., Telnet, HTTP) to outdated software with known vulnerabilities and public exploits.

<sup>2</sup>For instance, at the time of writing, Raptor has an estimated 22700 LoC, and SSL-proxy with Golang has an estimated 8600 LoC. Raptor and Golang are two of the frameworks we systematically retrofit.

<sup>3</sup><https://blog.yossarian.net/2021/02/28/Weird-architectures-werent-support-ed-to-begin-with>

From the attackers' point of view, IoT devices are very attractive due to the weaknesses they present and their absence of IoT-centered defensive tools (e.g., Antivirus, Intrusion Detection System (IDS)). Mirai is a proof of this. Mirai is the first malware specifically designed to infect IoT, which managed to infect around 600,000 devices [30]. Unlike the first samples of Mirai, which used only a set of usernames and passwords to gain access to IoT devices via insecure Telnet and SSH configuration, IoT malware currently incorporates a wide portfolio of exploits for N-days vulnerabilities in order to gain access, install and spread their malware [31]. Particularly, in a vast number of cases they (ab)use CVEs for web-interfaces [32]–[34].

IoT devices often provide network management interfaces for their configuration and maintenance (i.e., Telnet, SSH, HTTP), due to the lack of interactive interfaces like the ones offered in desktop computers (i.e., mouse, keyboard, video). Consequently, these network services are exposed to the attackers, offering access to the devices through notorious security issues, as shown in [2], [3], [35]. In [2], authors perform a large-scale analysis of web services provided by different IoT devices, discovering 225 high-impact vulnerabilities (i.e., Command execution, Cross-site scripting) verified through dynamic analysis, and around 9000 possible vulnerabilities reported through static analysis in 185 firmware images that were analyzed. These security issues, coupled with the shortage of security updates or patches, make IoT devices an attractive target for attackers, allowing miscreants to create large botnets or to mine cryptocurrency [36].

The next section introduces *HALE-IoT*, a methodology designed to offer an extra layer of security to IoT devices, focusing on the protection of the core and yet the most vulnerable services exposed to the Internet by IoT devices.

### III. HALE-IoT

This section introduces the main components of our system.

#### A. Methodology & System Overview

*HALE-IoT* is a system designed to harden IoT devices. Motivated by the wide-spread network of obsolete IoT devices, our system focuses on legacy and End-Of-Life (EOL) firmware for the following reasons: i) devices with a legacy firmware are prone to be vulnerable; ii) they are extremely likely to expose exploitable services leading directly to breaches; iii) and almost certainly they will never get a security patch. For example, in Figure 1 we depict a high-level view of a hardened IoT firmware/device. From the point of view of “inner components” of a firmware, the diagram captures two dimensions: 1) “vulnerable services” – the services that will eventually be spawned when loading the firmware (e.g., a web HTTP server, TELNET, FTP); and 2) “file system” – the original file system itself, i.e., where the binaries and the configuration of such potentially vulnerable services reside. The diagram also represents the entry point of these services, typically a network interface. A service can listen to different interfaces at the same time (e.g., Wireless or Wide/Local Area

Network, WLAN and WAN/LAN respectively). We represent as 0.0.0.0 a generic network interface which is very likely to be attacker-accessible.

*Methodology:* We follow three core principles when designing the methodology that underpins the development of *HALE-IoT*. First, the hardening process has to be generic and flexible to accommodate the most popular services available in IoT devices. We also require that the system can accept the integration of generic protection mechanisms that match in complexity the type of attacks that generally target IoT devices. Second, we follow the fail-safe minimization principle [37] by which the modifications we introduce during the hardening process should be as least intrusive as possible, always preserving the normal-operation of the device. In other words, *HALE-IoT* will perform minimal changes to the firmware, having its main focus to first harden the system via re-configuration, then patch existing *configuration files*, and only proceed to making code-level modifications (namely, binary patching) as a last resort. Only in situations when binary patching is necessary, we apply a two-fold strategy: the analysis phase — a human-guided semi-automatic process that produces a proof-of-concept; and the deployment phase — which can reproduce the patching and retrofitting at scale in a fully automated fashion.

*System design:* *HALE-IoT* leverages the methodology above to design a practical system that addresses the challenge of hardening heterogeneous devices from the following angles:

- **Secure front-end.** This step aims at hardening insecure services through the deployment of wrapper(s) designed to turn a possibly vulnerable service into a secure one. *HALE-IoT* will expose a secure interface of the service and it will act as a proxy of the actual service while offering certain guarantees, such as confidentiality and secured access control. Central to this step is the retrofit of a Secure Sockets Layer (SSL) proxy that will: 1) offer a cryptographic upgrade if the device lacks it, including the use of HTTPS instead of HTTP, SFTP instead of FTP, or SSH instead of TELNET; and 2) offer protection against SSL/TLS attacks (e.g., downgrade, MITM — Man In The Middle), and patch weak SSL/TLS configurations (e.g., hardcoded self-signed certificates).
- **Proactive detection.** This step aims at offering a proactive protection against application-layer attacks through the retrofit of a domain-specific firewall. For instance, *HALE-IoT* will implant a web application firewall when an IoT device processes web HTTP connections either directly from the user through a web browser or a RESTful client.
- **Advanced-level access.** This step aims at hardening a critical component of IoT devices, their admin interface. IoT devices do not generally have a graphical user interface, and their administration is generally done remotely.

The result of applying our methodology to hardening a generic IoT device is presented in Figure 1. In this paper, we offer an implementation of *HALE-IoT* that can scale the deployment of pre-hardened images for vulnerable legacy firmware that can benefit from a secure front end. We assume that these IoT devices expose services through the network while listening to a port through a socket. Our system performs best when there

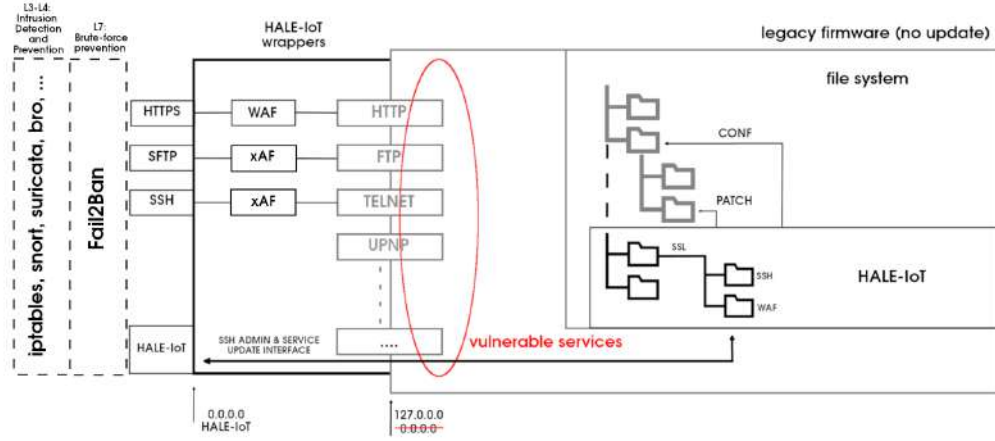


Fig. 1. IoT firmware system diagram AFTER hardening with *HALE-IoT* (shows also long-term vision for additional hardening at layers L3-L4, L7).

is a configuration file that specifies the network settings, and we restrict binary patching only to changes in the interface or the port number when these are hard-coded into the binary (see Section VII-F). Note that more intrusive modifications are subject to less automation, and it is thus less scalable and cost-effective. Also, the more intrusive modifications are highly likely to interfere with normal intended operation of the given service or even the entire device. While our methodology supports any type of binary patching, the assessment on the impact they have to the fail-safe minimization principle is the scope of our future work. We next describe in detail each of the three layers that constitute *HALE-IoT*.

## B. Secure Front-End: SSL/TLS Hardening

An SSL/TLS Proxy is a specific type of proxy server design to add a layer of SSL/TLS to protocols that lack this feature. For example, it is commonly used for adding HTTPS encryption to plain-text HTTP services without native HTTPS support. It is mainly responsible for the encryption and decryption of SSL traffic between the client and the server, and redirecting the packets once decrypted to the HTTP web server. As mentioned, the rationale behind adding an SSL/TLS proxy is driven by the prevalence of IoT devices running insecure or weakly secured HTTP implementations [26]–[28]. This in turn provides a *false sense of security*.

In our implementation we used two approaches for SSL/TLS proxying – SSL-proxy [38] as the main approach, and Lighttpd [39] as an alternative approach. SSL-proxy is a project written in the Golang programming language. SSL-proxy features a high portability to other systems, making it a good candidate for systems that require multiple architectures. SSL-proxy allows generating self-signed certificates as well as working with existing certificates and full certificate chains,

that are stored locally or generated through Let’s Encrypt [40]. For SSL-proxy cases, our toolsets generate Go binaries for the different architectures *HALE-IoT* supports, and then use the same SSL-proxy code in corresponding interpreted environments. As an alternative to SSL-proxy, we cross-compiled a statically-linked version of lighttpd with SSL/TLS and proxy support, which similarly to SSL-proxy supports self-signed certificates or certificates generated through Let’s Encrypt. In practice, it was only necessary to use lighttpd-based TLS proxy for the real device presented in Section VI-D, however this lighttpd-based setup was also successfully tested on other several devices. For the purpose of our experiments we used self-signed certificates, but we later discuss deployment issues in Section VII-G. However, the main idea of adding SSL-proxy is to provide any IoT device a guaranteed and uniform way of HTTPS support (e.g., latest TLS protocols) that can also operate proper full certificate chains [26]–[28].

## C. Proactive Detection: Application-layer Firewall Hardening

A generic Application-layer Firewall (xAF) is the type of firewall that can potentially detect and prevent malicious inputs designed to exploit specific application protocols. Our architecture allows the retrofit of multiple xAF, one for every potentially vulnerable network service. Therefore, *HALE-IoT* can both isolate local networks (from 0.0.0.0 to 127.0.0.1), and harden traditionally vulnerable services such as TELNET (secured with SSH), FTP (secured with SFTP), UPnP, MQTT. This architectural vision is presented in Figure 1.

Our current implementation of *HALE-IoT* methodology primarily offers support to harden web services at the application level. A Web Application Firewall (WAF) is an additional security layer that inspects web requests before redirecting them to their destination, allowing it to detect potentially malicious

requests and avoid redirecting them to the web server or to the web application. When a malicious request is detected, the WAF is supposed to prevent the request from reaching the web server, being able to detect the most common attacks at the web application level such as Structured Query Language (SQLi), Remote Code Execution (RCE) injections, Cross-Site Scripting (XSS), or Cross-Site Request Forgery (CSRF) attacks. In particular, we use *Raptor* [29] which is a lightweight open-source WAF written in the C programming language. It has very few dependencies, making it a good candidate for use in embedded systems. *Raptor* adds an additional security layer protecting web applications by comparing the content of HTTP requests with common signatures using a Deterministic Finite Automata (DFA) algorithm. Additionally, its functionality can be extended with rules and other matching strings algorithms, such as Karpe Rabin, Boyer Moore Horspool, or Perl Compatible Regular Expressions (PCRE). We cross-compile *Raptor* for the MIPS64, MIPS32, ARMv7, and Intel 80386 architectures, which are the ones currently supported by *HALE-IoT*. However, there is virtually no limitation to which CPU platforms *Raptor* (or any other WAF) can be cross-compiled for. At the same time, *HALE-IoT* could implant any other WAF as long as it can be either cross-compiled to native binary format for device's CPU, or can run in a cross-compiled runtime environment (e.g., Python, Go). The only unavoidable limitation our system inherits stems from the constraints of the actual devices in terms of obsolescence of runtime, RAM memory, and flash storage (cf. Section VII-E, VII-H).

#### D. Administration of *HALE-IoT*

*HALE-IoT* is composed of third-party components (e.g., WAF, xAF, HTTPS proxy) that may require bug-fixes, improvements, and configuration updates over time. For example, there is also a constant evolution of the threat landscape (e.g., applicable vulnerabilities, working exploits) against which *HALE-IoT* offers protection to the (legacy) IoT devices, and as such requiring updates to the xAF rules-set. These and similar related factors dictate the need of a way to administer *HALE-IoT* in an easy, secure, universal, and low-footprint manner. A classical way would be to use a web-interface to administer *HALE-IoT*, however we opt for an SSH-based administration.

There are several reasons why we choose the use of an SSH-based interface for *HALE-IoT* administration, instead of a web-based administration interface for example. First, SSH by default has a proven and strong built-in authentication and authorization mechanism and protocol based on public-private key infrastructure. In case of web-servers, it would require adding HTTP and/or HTML authentication models that would add to the complexity of implementation and maintenance, as well as potentially expose own set of authentication/authorization vulnerabilities. Second, compared to a web-interface, SSH does not require additional third-party dependencies and interpreters (e.g., PHP, Python) to provide full-fledged server-side functionality. With SSH-based approach the overall "application attacks surface" remains generally the same even after adding the new SSH dependency. The web-interface option on the other hand would increase the attack

surface through addition of the web-server and the admin web-pages themselves. Third, SSH provides a simple yet powerful interface to perform additional system-administration tasks should the need require (e.g., reboot, power-off, filesystem access). In case of a web-interface, there could be certain limitations to what administrative actions would be available to the web-server or the web-pages. Last but not least, efficient SSH implementations can be statically built with much lower footprint and overhead (e.g., Dropbear SSH at 100–200 KB). Such footprints are considerably lower when compared to most web-servers coupled with runtime interpreters (e.g., PHP).

#### E. Other Types of Hardening

The simplicity and flexibility of *HALE-IoT* approach is one of its core design principles (as stated in Section III-A) which also makes this one of its strong aspects compared to the state-of-the-art. In this sense, an additional improvement by *HALE-IoT* would be the addition of hardening at networking layers L3-4 and L7. In essence, it would mean protecting all the interfaces and all the services in a generic whole-system manner against network layer attacks (L3-4) as well as against application layer brute-force attacks (L7). For layers L3-4, *HALE-IoT*'s architecture can integrate industry-standard tools like iptables, Snort, Suricata, Bro, and for layer L7 fail2ban.

There are several adoption challenges that need to be considered. First, some IoT devices may not expose direct or standard access to various interfaces, thus requiring more intrusive re-configuration, binary-patching, or OS/kernel "hacks". Second, some IoT devices may use less common OS flavours (i.e., other than Linux-derivatives), hence re-binding and configuration of network interfaces may be different and may require certain *HALE-IoT* implementation adaptations. We thus leave the implementation and evaluation of additional L3-L4 and L7 retrofits as immediate future work.

### IV. EXPERIMENTAL SETUP

To evaluate the effectiveness and efficiency of *HALE-IoT* method, we applied it to web services of a wide-range of IoT devices. We chose to harden and evaluate web services as immediate focus, because these are the most commonly present services on most IoT devices. For this, we retrofitted and emulated 395 firmware packages with defensive implants containing a WAF (for prevention of web-attack vectors), and an HTTPS-proxy (for proper end-to-end HTTPS support).

In order to implant *HALE-IoT*, we have identified the web server configuration files and have re-configured them for hardening as follows. First, taking into consideration firmware's CPU architecture, we copy corresponding cross-compiled files to the firmware filesystem the implants needed by *HALE-IoT*. This includes executable and other files for the hardening elements (*Raptor*, SSL-proxy, Dropbear), configuration and rules-set files, and authentication keys for *HALE-IoT* SSH sysadmin interface. Then, we add the initialization scripts of the tools to the set of scripts that will be executed once the booting process finishes (e.g., init.d, rc.d, registration.d). Finally, we re-configure the web server configuration files or web server initialization scripts to isolate the interface and



TABLE I  
OUR INITIAL FIRMWARE DATASET (BY VENDOR AND ARCHITECTURE)

Vendor	ARMel	MIPSel	MIPSeb	Intel/386	Total	HALE-IoT
Asus	0	0	1	0	1	0
Belkin	4	10	30	0	44	0
Buffalo	1	0	2	0	3	0
D-Link	76	54	1	0	131	37
Huawei	0	0	1	0	1	0
Linksys	2	3	19	0	24	8
Netgear	45	174	150	0	369	46
OpenWrt	1	146	166	7	320	293
Tenvis	0	0	6	0	6	0
Tomato by Shibby	8	0	171	0	179	0
TP-Link	2	152	5	0	159	0
Trendnet	3	25	9	0	37	11
Ubiquiti	8	3	0	0	11	0
<b>Total</b>	<b>150</b>	<b>567</b>	<b>561</b>	<b>7</b>	<b>1,285</b>	<b>395</b>

TABLE II  
DISTRIBUTION OF THE INITIAL FIRMWARE DATASET (BY WEB SERVER AND CONFIGURATION FILES).

Web server	# of FWs	# of FWs (config file)	# of FWs (HALE-IoT)
lighttpd	58	52	42
httpd	649	74	68
minihttpd	37	0	0
AppDemo	71	0	0
boa	46	44	7
uhttpd	390	314	278
webs	2	0	0
goahead	9	0	0
Not found	23	0	0
<b>Total</b>	<b>1,285</b>	<b>484</b>	<b>395</b>

listening port of the service (e.g., original web server re-bind to 127.0.0.1 : 81), and then we start full-system firmware emulation [2], [3]. This process has been *fully automated*, and has been carried out for each of the tests of the evaluation.

## A. Dataset

Our initial dataset consists of 4,809 real-world firmware images extracted from FIRMADYNE [3]. Note that the original FIRMADYNE dataset is larger, but 4,809 images are available to download at the time of writing. We then retain only the images in the architectures that *HALE-IoT* currently supports (i.e., ARMel, MIPSel, MIPSeb, and Intel 80386 cf. Section III), making a total of 1,328. From these, we discard 43 images that have a custom format compression algorithm hence cannot be systematically unpacked with Binwalk [41] (which comes as part of FIRMADYNE setup). After processing all remaining images, we manage to extract the root filesystem from 13 device vendors (ranging from Asus to Ubiquiti) for 1,285 images overall. Overall, these root filesystem are associated with devices of the following type: Ethernet routers, WiFi routers, xDSL modems, IP cameras. Table I shows the distribution of vendors in our dataset, per CPU architecture.

It is important to note that when trying to address such an immense and heterogeneous experimental population and space, for practical and resource reasons we are bound within magnitudes that are feasible for handling such experiments. At the same our work exceeds comparable experimental state of the art works such as FIRMALICE [42] (dataset size: 3 samples), and generally positions our experiments within the magnitude range of similar state of the art works such as FIRMADYNE [3] (dataset size:  $N \times \text{pow}(10, 2)$ , i.e., hundreds of samples).

## B. Emulation

To evaluate *HALE-IoT* we emulate a device that runs the firmware images in our dataset. Since we are mainly interested in systems that have a Web interface to administrate the device, we next describe the steps we take to select those images. We first scrape the file system of the image in the look for binaries that are core components of a Web server (e.g., uhttpd). We then identify the configuration files that inform settings to the Web server (e.g., boa.conf, lighttpd.conf). Table II shows the different type of web servers together with the number of firmware images (marked as “FWs”). As expected a large

subset of images have a web server configuration file together with the server binary, an exception being uhttpd. Images with a binary and without a configuration file have the settings of the server embedded in the binary itself. To scale our evaluation, we focus primarily on the 484 images that have an explicit and non-embedded configuration file.

We note that from architecture and design perspectives, *HALE-IoT* can run virtually on any type of firmware as long as the user(s) can change the binding network interface and port of the service that we aim to harden. However, in certain cases (e.g., service uses custom or binary-hardcoded configuration) changing the network interface and the port may require more manual effort, and we discuss such challenges in Section VII. In the end, out of all 484 images that have a Web server configuration file, we manage to successfully emulate and implant *HALE-IoT* to 395 firmware images. The emulation and *HALE-IoT* implant covers the following five vendors (D-Link, Netgear, TRENDnet, Linksys, and OpenWrt) (Table I), and the following four web-servers (lighttpd, httpd<sup>4</sup>, boa, and uhttpd) (Table II). Once the emulation started, we were able to successfully communicate with all 395 web server processes and, more importantly, we were able to retrofit the *HALE-IoT* security hardening measures in all those firmware images.

## C. Toolsets

One key aspect of *HALE-IoT* is that it supports out-of-the-box different CPU architectures, and is flexible enough to keep adding more architectures and defenses in the future. In particular, we compile our framework for ARMel, MIPSel, MIPSeb, and Intel 80386 as discussed before. While it is possible to use QEMU to emulate one-by-one the operating system used by the different vendors, we opt to perform a systematic cross-compilation through a toolchain. There are different toolchains available including Linaro [43], or Linux MIPS Toolchain [44]. For the purpose of this paper, we create our own customized toolchain using *Buildroot* [45]. Our toolchain uses *musl* [46], which implements the standard C library with some improvements such as enhanced support for static linking. When cross-compiling the different binaries using our toolchain, we strip the binary of all symbols to optimize size. At present, we automatically cross-compile *Raptor* (for WAF),

<sup>4</sup>In most IoT devices we encounter, “httpd” is just a generic placeholder name for the web-server, and should not be assumed Apache’s HTTP server.

and *Dropbear* (for *HALE-IoT* SSH-based administration) for all the supported architectures. We do not cross-compile SSL-proxy, as it is written in Go programming language and the binaries for the different architectures can be generated directly without using a specific toolchain. Final builds of the toolsets resulted in the following footprints: Raptor 275.8–346.6 KB, SSL-proxy 5053.5–6244.3 KB, Dropbear SSH 179.8–228 KB and *lighttpd* 2381–3018 KB. For additional resource overheads incurred from *HALE-IoT* implant, see Section V-E.

It is important to note that our experimental setup is systematic and is easily extensible to other architectures and defensive toolsets, which is precisely the scope of our future work as discussed in Section IX.

## V. TEST METHODOLOGY AND RESULTS

### A. Test Methodology

We run two tests for each QEMU-emulated firmware following a DevOps methodology [47], [48]. This methodology evaluates changes into a system in an incremental fashion so that failure causality can be properly attributed. One test contains a hundred common (i.e., non-exploiting) web requests, while the other test has a hundred web requests with some type of web-attack payload (e.g., XSS, SQLi, Command Injection).

Our DevOps-style testing methodology has the following steps. First, we emulate the firmware without any kind of modification and we run the tests as a control measure to evaluate the differences. We also check how many firmware images accept connections through the HTTPS. Then, we retrofit the firmware with the Raptor WAF and we launch both tests again. We do the same again but only after implanting the SQL-proxy in the firmware. Finally, we launch both tests on the firmware emulated with both protection measures retrofitted, that is, the Raptor WAF and the SQL-proxy working together. We also ensured that random non-malicious requests return exactly the same result in both tests (i.e., with and without *HALE-IoT*). For this, we compare the HTTP headers and the content returned in both test setups (normal vs. non-malicious) requests. To assess the performance of the devices after retrofitting the WAF in a realistic setting, we use Raptor’s DFA algorithm and fifty-five regular Perl Compatible Regular Expressions (PCRE) we gather from the community [49], [50].

We apply our test methodology to answer the following questions: *Q1) Is HALE-IoT able to retrofit defensive firmware modifications and implants without disrupting the normal operation intended for the firmware?* *Q2) Can HALE-IoT effectively deploy a secure front-end in legacy devices?* *Q3) Can HALE-IoT effectively deter known attacks and known vulnerabilities against legacy devices?* *Q4) Can HALE-IoT effectively cover multidimensional heterogeneity (e.g., physical vs. emulation, ARM vs. MIPS vs. x86, real CVEs vs. synthetic vectors, open-source vs. proprietary, cross-vendor)?* *Q5) What is the potential performance overhead incurred by HALE-IoT?* *Q6) Can HALE-IoT actually work with services that apparently cannot run on 127.0.0.1 : < port > via configuration file?* *Q7) Finally, does HALE-IoT actually work on physical devices, rather than just emulated environments?*

We evaluate question Q5 using emulation only, question Q7 using bare-metal hardware only, and questions Q1, Q2, Q3, Q4, and Q6 using both bare-metal and emulation.

### B. Correctness and effectiveness of the Retrofit (Q1)

To test how well *HALE-IoT* retrofits defensive firmware modifications, we deploy all hardened images in our own emulator environment resembling the one in [2] and borrowing additions from *FIRMADYNE* (e.g., NVRAM) [3]. Table III shows a summary of our results. We see that all 395 images remain functional, i.e., the hardening process does not disrupt the normal operation intended for the firmware. However, some cases under-perform in terms of isolation. In particular, we see that in 8% of the images we continue to see the web server listening in the external interface (0.0.0.0), and thus potentially-exploitable connections are possible to the original web server without going through our hardening proxy-chain. We also see that 78% of the emulated firmwares do not use HTTPS by default before the retrofit. We next explore in detail the performance of *HALE-IoT* when looking at the first layer of its architecture (cf. III-B).

During our *HALE-IoT* experiments neither our human experts nor our automated tools have encountered any functional abnormality, and the emulated-and-hardened web services along with the entire system emulation performed normally and as expected.

### C. Secure front-end in legacy devices (Q2)

One of the main hardening goals of *HALE-IoT* is to isolate the vulnerable services from attacker-accessible interfaces (e.g., WAN, LAN), while at the same time keeping the original services running on 127.0.0.1 to satisfy Q1. Our evaluation shows that *HALE-IoT* successfully re-configures original web-servers from 0.0.0.0 : 80 into 127.0.0.1 : 81, replacing the former address with the service running our WAF implant while relaying only safe HTTP web requests to the original web-server now residing in the latter address.<sup>5</sup>

While further analyzing the results shown in Table III, we make two key observations. First, 100% of the original web-servers (from the successfully emulated 395) re-bind well to port 81 as instructed by *HALE-IoT*’s re-configuration routines. Second, despite being explicitly instructed to change binding from 0.0.0.0 to 127.0.0.1, there are 32 firmware images that remain bounded to 0.0.0.0 (in addition to the new address). This can keep exposing a potentially vulnerable service to attacker-accessible interfaces, thus rendering our hardening ineffective. We posit this is due to vendor’s (un)intentional implementation and coding choices or errors, where only some values from the configuration file(s) are taken into consideration while the rest of parameters are either hardcoded into the binary executable or taken from other non-obvious configuration files. We present some evaluation of *HALE-IoT* for this case in Section VI-A, and discuss this challenge in

<sup>5</sup>When WAF is chained with SSL-proxy, the WAF is further isolated to 127.0.0.1:80, and web service is exposed by SSL-proxy binding to 0.0.0.0:443.

more detail in Section VII, but we emphasize that this happens only in 8% of our images.

Another of our aims is to add secure tunnel wrappers around services through the use of SSL-proxy. Our rationale is that these services either have weak secure tunnels, or are just plain-text altogether (i.e., adding HTTPS support to IoT devices that quite commonly are lacking it). As discussed, in certain instances the IoT device may provide by default an HTTPS server. In 84 emulated firmware images, the original web servers also start a “default HTTPS server”. However, besides carrying a self-signed certificate, the “default HTTPS server” also featured an outdated SSL/TLS version (e.g., TLSv1), hence still very likely exposing the web interface to various HTTPS and MITM attacks. On the other hand, with *HALE-IoT* (e.g., with SSL-proxy) we are able to provide the hardened IoT devices with the latest and most secure TLS implementations along with the proper support for full certificate chains (see also discussion in Section VII-G). This in turn provides real increased security rather than merely a “sense of security” provided by most *default HTTPS servers* when these are implemented in IoT devices and working with self-signed or expired certificates.

Overall *HALE-IoT* automatically manages to fully isolate 92% of the potentially vulnerable web services, while correctly providing a secured SSL tunnel in 100% of the tested cases.

TABLE III  
CORRECTNESS AND EFFECTIVENESS OF THE RETROFIT 395 EMULATED WEB SERVERS.

Test under evaluation	# of emulated FWs	(%)
Firmware is <i>functional</i> after retrofit	395	(100%)
Isolation OK (web service re-bind to 127.0.0.1)	363	(91.9%)
Isolation FAIL (web service hardcoded to 0.0.0.0)	32	(8.1%)
Default HTTPS present before retrofit	84	(21.3%)
Default HTTPS <i>missing</i> before retrofit	311	(78.7%)
HTTPS present <i>after</i> retrofit	395	(100%)

## D. Detection and prevention of attacks and exploits (Q1, Q3)

To evaluate the performance of *HALE-IoT* in regard to its second architectural layer (cf. Section III-C), we perform two experiments.

**Automated-attacks.** We leverage a battery 200 web requests, out of which half are common requests and the other half are known web attacks. The attacks include known XSS, SQLi, and Command Injection attacks coming from both actionable CVEs and synthetic input. Our results show that *HALE-IoT* can detect all known attacks when configuring the WAF vanilla community detection rules. The detection rate itself is not at all surprising, but this experiment reports a valuable finding: *HALE-IoT* can reliably retrofit complex defense mechanisms into the firmware of IoT devices through binary retrofits while keeping the original firmware functional (Q1), and offering the full-fledged level of protection of the retrofitted secure mechanism (Q3).

**Targeted-evaluation.** We also evaluate the effectiveness of *HALE-IoT* by targeting some firmware images with CVE-2016-1555 (also known as ACSA-2015-001). CVE-2016-1555 was independently discovered by Chen et al. [3] and Costin

et al. [2]. This known vulnerability covers a series of pre-authentication XSS and RCE in several devices from Netgear (many of which are already EOL, hence will remain unprotected indefinitely unless it gets hardened with *HALE-IoT* or similar). First, we exploit the vulnerabilities in the emulated environment and confirm that the original firmware is vulnerable and exploitable. Then, we apply *HALE-IoT* to the emulated firmware and see that all attacks are efficiently stopped. This further proves the effectiveness of our approach, but this time with an attacks that targets End-Of-Life devices. We refer the reader to Appendix A for visual representation of the success of our proof-of-concept attack and defense.

**Takeaway.** *HALE-IoT* can effectively deter known attacks against legacy devices. Naturally, our system inherits the limitations of the defense mechanism we implant. In particular, Raptor is mainly effective at detecting known attacks and can miss connections that encapsulate zero-day web attacks. We discuss this limitation in detail Section VII through different axes, including the WAF’s inherent limitations (§VII), and limitations in the datasets (§VII-C). However, we also note that the overall effectiveness of *HALE-IoT* when it comes to the detection of attacks has to be seen from an holistic perspective. In Section V-C we report the effectiveness of our system at hardening insecure (superfluous) services other than HTTP. When putting together the secure front-end and the proactive detection layers (Figure 1), *HALE-IoT* can offer a system resilient to both known attacks against web services, and against unknown attacks targeting all other hardened services.

With *HALE-IoT* implanted, we achieve 100% detection and prevention rate of known attacks in both emulated and real-devices, while effectively hardening other services that are often target of unknown (zero-day) attacks. This 100% detection ratio is taken as a unit test rather than a detection ratio. This provides assurances that the WAF we retrofit works as expected under active attacks. We are aware that WAF systems detect attacks for which there is a known rule, and they are unquestionably subject to evasion, just like any other rule-based detection system—whether the WAF is on a high-end production system, VPN appliance, or a HALE-IoT retrofitted router/camera. However, they provide an extra layer of security that protects against known exploits targeting the firmware’s web interface, and they prevent most automated attacks (i.e., via bots looking for vulnerable devices) that target vulnerabilities in the exposed web servers of IoT devices [5], [35].

## E. Functional, Performance, Overheads Evaluation (Q4, Q5)

We have collected measurements of the performance overheads introduced by various components of the *HALE-IoT* implant. Since *HALE-IoT* is highly flexible and configurable, we use a modular analysis to assess our performance. This is, we measure the performance of the WAF alone, the SSL-proxy alone, and the SSL-proxy chained with WAF. For each test we collect benchmarks for the CPU and memory consumption, as well as the response time of the web requests. The performance evaluation in the emulation provides an approximation of the memory and CPU consumption as

the difference between runs with and without any type of retrofitted tool.

In Figures 2, 3, 4 the references to “WAF”, “SSL-proxy”, and “WAF + SSL-proxy” represent the use of *HALE-IoT* with a particular self-descriptive configuration. On the other hand, “Control” represents firmware emulation without any added components. As we discussed in Section V, each test is made up of one hundred common web requests represented as “Normal”, and one hundred requests that contain some type of attack represented as “Crafted”. We carry out all tests in each of the firmware that we emulate and implant *HALE-IoT* into. In total, our evaluation scripts made 316,000 web requests. Figures 2, 3, 4 represent the average of the results over the entire set of emulated and tested firmware images. The data has been collected using common Linux tools (e.g., *mpstat* or *vmstat*) from the host side.

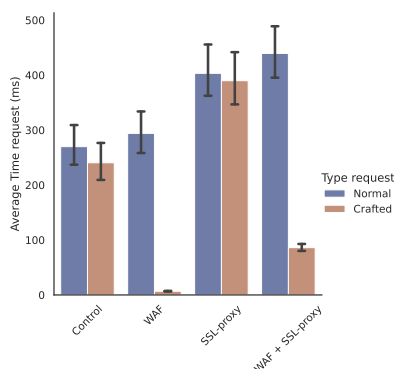


Fig. 2. Average time to complete the HTTP request for each test run.

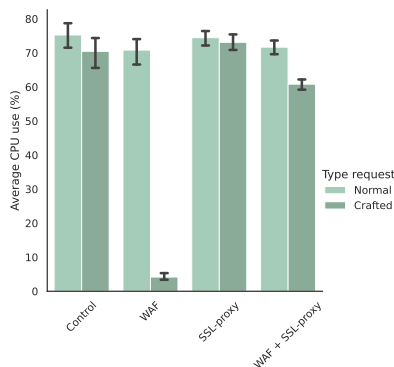


Fig. 3. Average CPU consumption for each test run.

We also provide an interpretation of the performance overhead graphs. In Figure 2, we see that the response time for “Normal” requests increases proportionally to the number of chained components for the particular hardened service. For example, “WAF + SSL-proxy” complete request time

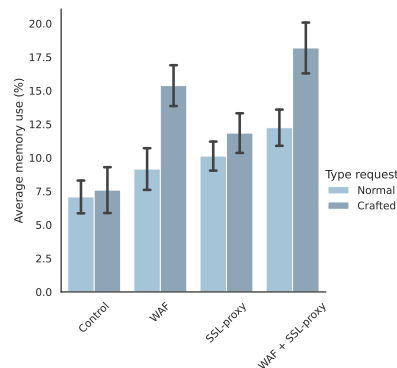


Fig. 4. Average RAM memory consumption for each test run.

takes longer than “WAF” or “SSL-proxy” separately. This is somewhat expected as the request in the case of “WAF + SSL-proxy” is being forward back and forth via multiple connections and software modules that have their own context-switching delays, etc. At the same time, Figure 2 shows that in case of “Crafted” requests, whenever the “WAF” component is present the complete request time is significantly lower compared to Normal. This is both expected and a direct evidence that the “WAF” effectively detects and blocks attack attempts, and as such protectively terminates at much earlier stages the HTTP communications carrying potentially malicious payloads. A similar pattern can be seen in Figure 3. In case of “Crafted” requests, whenever the “WAF” component is present (e.g., when only “WAF” is present without “SSL-proxy”), the average CPU usage is sensibly lower compared to “Normal”. Once again, this is both expected and a direct evidence that the “WAF” effectively detects and blocks attack attempts, as “WAF” does not continue any further computations and processing (e.g., relaying it to the original web service) once it has detected and prevented potentially malicious payloads.

Moreover, the average memory consumption increase shown in Figure 4 is expected as the additional components require memory for operation and for storing their data such as WAF rules-sets and SSL/TLS certificate chains. The memory consumption, though, is harder to fully interpret when compared to CPU usage and processing time of requests, as the coding practices can vary greatly across the applications. Also, as opposed to CPU usage which stops when a particular function flow stops (e.g., HTTP request blocked and terminated), the memory often is not immediately freed (or not made visibly available to OS, even if freed by the application) when the program reach certain states such as “Crafted” packet detected and blocked. In terms of storage overhead introduced by *HALE-IoT*, Figure 5 shows the distribution of sizes for all firmware images along with the retrofitted binaries. Specifically, as detailed in Section IV-C, the increase by each components is as follows: Raptor 275.8–346.6 KB, SSL-proxy 5053.5–6244.3 KB, Dropbear SSH 179.8–228 KB.

During the evaluations presented in this work, we did

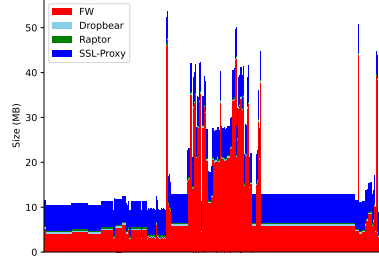


Fig. 5. Size of the FWs along with the size of the retrofitted binaries.

not particularly focus on performing an *exhaustive regression testing* on the web interface (nor on other functions and services) operating within the evaluated devices and emulated firmware. Because such exhaustive complete system regression testing would be a non-trivial experiment in itself, we leave as future work the large-scale evaluation of the functional impacts induced by retrofit defense systems such as *HALE-IoT*. However, we have also performed an evaluation of the retrofitted software. We apply fuzzing by creating a harness for the main core of the WAF (i.e., the function to which any data received through the socket is passed) and use American Fuzzy Lop (AFL) as a fuzzing tool. We found some memory related bugs and several crashes in the fuzzing stage that we patched. The crashes occurred in *wafmode* four (one of the command line options). This parameter has four levels of protection, number four being the highest, and defines the mode of the DFA algorithm to detect common attacks. With the rest of the modes and with DFA disabled using only regular expressions as rules, the application did not produce any crashes with the same test cases. After a further inspection and debugging, the crash occurs when trying to read a value beyond the stack limit, which causes a segfault. This error is caused by the use-after-return memory error, and since this memory area belongs to a function that has already terminated, it can cause undesirable behavior. We discuss the bugs found as well as the security limitations that any retrofitted piece of software may have on a system in Section VII-I.

A differentiating end goal of *HALE-IoT* w.r.t. the related work (e.g., ABSR and Symbiotes [51]) is that we aim at being as least intrusive as possible, and to ensure that legitimate requests do not have in important impact in the performance of the device. Our results strongly support this goal.

We see that the use of *HALE-IoT* introduces some interesting trade-offs. When attacks are blocked, we effectively reduce the overhead. Judging by the performance of the hardened device when processing legitimate requests alone, we see that *HALE-IoT* does not introduce an important overhead.

## VI. CASE STUDIES

We next present a number of case studies that aim at better understanding the performance of *HALE-IoT* in detail. In particular, we look at a firmware image from *Linksys*, and *Asus*.

TABLE IV  
SUMMARY OF REAL DEVICES PERFORMING SUCCESSFULLY IN OUR EVALUATION.

Device	CPU / Cores / MHz	Architecture	RAM	Storage
Asus RT-N12+ B1	MT7628NN / 1 / 580	MIPSel	32 MB	8 MB
Netgear R6220	MT7621ST / 1 / 880	MIPSel	128 MB	128 MB
Linksys EA4500	88F6282 / 1 / 1200	ARM	128 MB	128 MB
RaspberryPi 3	BCM2837B0 / 4 / 1400	ARM	1024 MB	4096 MB

We conclude our case study with the deployment of a hardened version of *Asus RT-N12+ B1*, *RPi3 OpenWrt*, *Netgear R6220* and *Linksys EA4500* over four different hardware devices. Table IV summarizes the technical specifications of the actual physical devices used in our evaluation.

### A. Reverse-Engineered Hardcoded Binary for Linksys wrt54gs (emulation) (Q6)

As presented in Section V-C, there were 32 emulated firmware that failed to isolate web-server via binding to 127.0.0.1. For unknown reasons the firmware kept the web service binding to 0.0.0.0. In order to demonstrate that *HALE-IoT* is also feasible, practical, and effective even when the re-configuration retrofitting fails, we attempted a minimal-effort manual reverse-engineering of one such web-server binary.

For this we chose the “*httpd BusyBox*” web server binary from OpenWrt firmware built for wrt54gs device by Linksys. Even though, “*httpd BusyBox*” is known to support “*-p*” option to change the binding interface and port (e.g., “*-p 127.0.0.1:81*”), in this particular case it was not supported or it did not work. We then investigated the potential reasons behind this fail. The wrt54gs firmware image has a non-stripped BusyBox binary that is dynamically linked, therefore our first approach was to look for HTTP functions to recognize the httpd BusyBox applet. Then, we identified the call to the *bind* function and check the parameters backwards. We found the *inet\_aton* function that converts a string IP address into binary form, and that it is using as a parameter the variable assigned from the “*-l*” command-line argument. Though this argument does not appear in the help menu of the httpd command, it allows to change the listening interface of that specific httpd binary. We leverage the hidden “*-l*” option to successfully run *HALE-IoT* in the wrt54gs firmware.

We can further generalize this *one-time manual effort* into *HALE-IoT*’s automation as follows. We can identify similar service-exposing binaries using, for example, Yara rules [52], or heuristics and matching based on op-code level or semantic code-similarity [53]. Similar binaries could relate to: i) the same device (but different firmware version), ii) the similar device models (from the same vendor), iii) or even distinct devices across vendors (e.g., “white label” products). The takeaway from this case study is that manual efforts can sometimes provide “intelligence” that can help to scale the hardening of images over very large number of similar firmware environments.

### B. Evaluation on RaspberryPi with OpenWrt (device) (Q7)

To evaluate *HALE-IoT* over a bare-metal device, we deploy an OpenWrt (LEDE 2017 build) into a RaspberryPi 3 device.

OpenWrt is the most popular vendors in our dataset that has firmware images for all our architectures. The LEDE 2017 build version of OpenWrt has a known XSS in its LuCI web interface.<sup>6</sup> Therefore, we first run a non-hardened OpenWrt firmware and we see that the vulnerability can be exploited in practice (see Figure 8 in Appendix A). We then hardened the same OpenWrt firmware with *HALE-IoT* and see that with the “WAF + SSL-proxy” configuration we can completely prevent the XSS attack, in addition to being able to add full HTTPS support (see Figure 9 in Appendix A). This case study indicates that *HALE-IoT* works as expected on bare-metal devices, but our next case study further dwells on this.

#### C. Evaluation on Asus RT-N12+ B1 (device) (Q6, Q7)

We evaluate *HALE-IoT* on another bare-metal device we have had access to, i.e.: Asus RT-N12+ B1. This device runs MIPS32 binaries, and in particular uses a custom httpd as its web server. This image requires a retrofit at the binary-level as the configuration parameters of the web server are hardcoded into the binary and can not be identified by *HALE-IoT* automatically. After reversing it, we see that the binary accepts three arguments: 1) the name of the interface whose IP address will be obtained through `SIOCGIFADDR` ioctl; 2) the port; and 3) a way to enable SSL connections. The device also runs and exposes TELNET and SSH services that we used for “live implanting” (§ VII-A).

As a result of our implant, we see how *HALE-IoT* spawns the WAF into the device, and how the web service is secured behind HTTPS while hardening all other services. We see that Raptor works as expected, detecting and preventing potentially malicious input test-vectors. However, we notice that the device periodically faced some resource limitations manifested as unavailability of RAM memory. Non-deterministically, when insufficient RAM is available for handling HTTPS/HTTP/network requests via *Hale-IoT* processing chain, the spawned process/thread (e.g., WAF, SSL-proxy) is killed by the OS/Kernel due to lack of sufficient memory blocks to allocate. This is a limitation rooted into a combination of technical factors such as the hardware runtime environment (i.e., device with very limited RAM), and the implementation choices (i.e., SSL-proxy executable size). However, this case study shows how our methodology can harden Asus RT-N12+ B1. In practice, for this type of device, a more lightweight defensive mechanism would have to be deployed in order to make the added defenses effective and usable. We further discuss the implications that drive the choices of the implants in Section VII-H. This case study shows that our generic methodology let us work with heterogeneous firmware images, and it works also on bare-metal devices and firmware.

#### D. Evaluation on Netgear R6220 (device) (Q6, Q7)

Next, we evaluate *HALE-IoT* on Netgear R6220. This device runs MIPS32 little-endian binaries, and in particular it uses the *mini\_httpd* web server. The web server does not

contain any configuration files, and the server options are configured through the command line. Therefore, we can change the listening port and interface via the server’s arguments. We retrofitted Raptor and SSL-proxy on the device through a Telnet server that can be enabled in debug mode. Raptor worked as expected, however SSL-proxy did not work due to Golang incompatibilities with older MIPS kernel versions<sup>7</sup>. As an alternative to SSL-proxy, we use a statically-linked cross-compiled lighttpd server with support for SSL/TLS and reverse proxy. We use a configuration that listens on port 443 with SSL enabled and redirects incoming requests to the WAF. As a result of using lighttpd as an alternative, we can see that the extra security layers added by *HALE-IoT* are working correctly. This case study shows us that our methodology is functional, flexible and independent of the type of tools used.

#### E. Evaluation on Linksys EA4500 (device) (Q6, Q7)

Finally, we evaluate *HALE-IoT* on the Linksys EA4500 device. This device runs ARMv5 and lighttpd binaries as a web server. The default firmware does not present any access to the command line, which poses a challenge to *HALE-IoT*. However, we find a workaround that shows how our system can be deployed through unconventional means. Linksys EA4500 allows a user to connect USB devices to the router to share files over the network. When a USB is plugged in, it is mounted in the `/tmp` folder of the device. If a folder named `packages` exists, it is symlinked directly to the `/opt` directory. Finally, whatever file is present in `/opt/etc/registration.d/`, it will be executed by the shell<sup>8</sup>. Therefore, we use this hack/vulnerability to add a statically-linked version of dropbear and *HALE-IoT* into the device’s *running firmware*. As a result of this implant, Raptor WAF and SSL-proxy work properly together with access to the device via SSH to update or modify its configuration.

## VII. CHALLENGES AND DISCUSSIONS

### A. Delivery of the Retrofitted Implants

Modifying an existing firmware is the first step in the delivery of an implant, and it can be done leveraging tools like Firmware-Mod-Kit (FMK) [54]. However, in certain cases implants are not easy to realize in practice. This happens, for instance, when the firmware update needs a digital signature or there is a cryptographic protection (e.g., strong and secured private key, correct implementation of validation). However, there are also vulnerabilities that allow flashing a non-certified or modified firmware into a device with these restrictions. Some of these vulnerabilities relate to forging digital signatures or bypassing digital signature verification. Giese [55] exploits a Domain Name System (DNS) redirect to trick Xiaomi Cloud to download modified firmware from a local server. Another example is when there is no firmware update available, except the original firmware running on the device. Finally, low level frameworks like Firmware-Mod-Kit may be

<sup>7</sup><https://github.com/golang/go/wiki/MinimumRequirements>

<sup>8</sup>Dan Walters: <https://web.archive.org/web/20120914060622/http://blog.danwalters.net/>

<sup>6</sup>More details here: <https://github.com/openwrt/luci/issues/1731>.

unable to support the specific firmware format that requires hardening. We next discuss alternative methods that *HALE-IoT* could deploy to circumvent this limitation. These methods revolve around the idea of making the implant directly into the device in runtime.

The first option is the use of network or serial interfaces (e.g., Joint Test Action Group – JTAG, Universal Asynchronous Receiver-Transmitter – UART) to access the built-in TELNET and SSH services via the bootloader or the OS prompt. Then, we can implant *HALE-IoT* using automation scripts over traditional OS sysadmin techniques as shown in the case study in Section VI-C. The second option is to exploit a known vulnerability in the running device, such as Remote Code Execution (RCE) or Command Injection (CI), to inject benign code and implant the *HALE-IoT*, for example as demonstrated for Linksys EA4500 (see Section VI-E). Naturally, *HALE-IoT* can then also patch those particular vulnerabilities so that they can not be abused further. Note that similar techniques have been used by both highly-competitive malicious botnets and vigilante IoT malware [56]. While this section discusses the challenges of modifying firmware (software), we next look at the issues behind dealing with the actual devices (hardware).

### B. Persistence of the process behind Retrofitting Implants

The IoT realm is heterogeneous, and the process of retrofitting additional security into these devices is highly technical task. One task that remains particularly challenging is keeping these retrofits persistent across reboots and power-offs. *HALE-IoT* is stored at the filesystem level (e.g., flash storage) to keep persistence. However, several factors can prevent *HALE-IoT* from being persistent, including: factory resets, firmware upgrades, forceful flash storage cleanup (e.g., SPI communication with flash chipset), or even protections from manufacturers (e.g., restricting partitions to *read-only*). In many cases, the implants (both benign and malicious) can survive the above “cleanup” scenarios by installing an implant component at the bootloader level, thus essentially acting as a boot-time rootkit. However, this is a challenging research area that requires further explorations and ethical considerations. Other specific protections from manufacturers can be overcome with case-by-case basis techniques. For example, the restriction of partitions to *read-only* could be overcome with repack and re-flash of the firmware (cf. Section VII-A).

### C. Dataset Size and Representativeness

In order to analyze, harden, and test our *HALE-IoT* system, a dataset of firmware is required. The vendor’s website is the premier choice to gather firmware. However, third-party websites also host firmware images. The most convenient way to acquire firmware online is via web-crawlers [1], [3]. However, harvesting a dataset through web-crawling has its limitations. On the one hand, the firmware that were once available online are many times being pulled offline by the vendor. This can threaten the reproducibility of the evaluation. For example, state-of-the-art projects, such as FIRMADYNE [3], face this problem and we see a gap from the time of their release to the

time of our experiments — many URLs and firmware are not available online anymore [1], hence limiting the experimental datasets from the start. Even if the crawlers can be updated to work with a redesigned website (which is tedious and does not scale in effort), they fundamentally can not be fixed to download a firmware file taken offline by the vendor.

On the other hand, some devices do not have firmware images available online. This could be due to the nature of the product or the age of the device. Pulling the firmware out of a device through Telnet or secure shell (SSH) connections is possible in some scenarios [57]. However, in many cases, memory dumps through hardware hacking is the only viable option [58], [59]. IoT devices often accommodate low-level hardware interfaces such as Universal Asynchronous Receiver/Transmitter (UART) or Joint Test Action Group (JTAG), through which it is possible to connect directly to the device’s bootloader or to its root shell [59], and then take a storage and memory snapshot, or just perform a live-implant. This approach requires extensive human expertise and interventions, and also does not scale well.

A fundamental challenge to all research targeting IoT devices and firmware the lack of a highly representative baseline of IoT firmware dataset. Building a dataset is challenging and tricky from multiple perspectives. On the one hand, collected firmware can face copyright scrutiny from vendors if it includes proprietary firmware. Also, it is highly unlikely that many relevant and omnipresent vendors will sign-off releases of their firmware into such a dataset. On the other hand, ignoring proprietary images and including only open-source firmware would be relatively easier to accomplish, however this would bias the dataset and make it unrepresentative toward the myriad of Commercial-Off-The-Shelf (COTS) devices running proprietary firmware. Our evaluation uses the FIRMADYNE dataset [3], which is considered the state-of-the-art. From it, we were able to successfully process 395 images, which it is comparable in size to the datasets used by prior works [2]. However, given the large number of IoT vendors, this dataset can be seen as limited. Future work is needed to create a dataset, that is:

- *highly representative* baseline of IoT firmwares (e.g., multi-dimensional representativeness — CPU arch, OS, device type, core services and functionality, networking interfaces and stacks, firmware packaging formats),
- *not facing licensing issues* (e.g., firmware that is proprietary, non-distributable, etc.),
- *stable and always available* for download, duplication, and improvement (e.g., never lost partially or totally).

### D. Firmware Obfuscation

Firmware images are often packaged and compiled, thus, requiring preparation before analysis [1]. Specialized software, such as Binwalk [41] and BinaryAnalysis-NG (BANG) [60], is used to unpack the firmware revealing the file system and other information, thus enabling further analysis. However, as there is no standardization, some manufacturers try to obfuscate and complicate unpacking and reversing their firmware, for example, by adding custom format compression [1]. Due to

the memory and other resource constraints, IoT devices often ship with file systems designed for constrained devices, such as Squash File System (SquashFS) or Journaling Flash File System (JFFS, JFFS2) [61]. These file systems are often read-only and have file system compression enabled. Additionally, software such as Firmware-Mod-Kit (FMK) [54] is one of the few available and one of the most popular tools to perform firmware modifications on a relatively wide range of formats and devices.

When not performing a live implant, *HALE-IoT* focuses on retrofitting legacy binary firmware via firmware modifications, hence it requires and performs: i) the unpacking, and modification steps (if emulation is involved); ii) the unpacking, modification, and also *repacking* (if physical devices force a firmware upgrade for implant to work). In these cases (especially when physical device with firmware upgrade is involved), in the end must produce a firmware that is accepted by the device and is fully functional. However, even though both firmware unpacking as well as modification-and-repacking are somewhat represented by existing toolsets, current state-of-the-art does not address the fundamental challenges of unpacking and modification-and-repacking of non-trivially obfuscated or encrypted/signed firmware. In this sense, *HALE-IoT* system inherits all the limitation of the existing tools (e.g., Binwalk, BANG, FMK), which however is not a limitation of the *HALE-IoT* methodology. In our current evaluation, the physical devices and the emulated firmwares were representative to the IoT device populations that allows relatively easy live-implanting as well as unpacking, modification, and repacking. We posit that more work is required to overcome the analysis of obfuscated or encrypted firmware packages.

#### E. Runtime Environments Being Obsolete

Runtime challenges became notorious when we started experimenting implants into random COTS IoT devices for the purpose of our case study. We next discuss some of those challenges to illustrate the complexity of the problem space and elicit research efforts towards better instrumenting obsolete runtime environments.

In one instance, the router undergoing hardening had the vendor's original firmware, and was running BusyBox, Linux kernel, and other executable files compiled for MIPS-I. However, the Buildroot environment we use (including many of its prior versions), while producing MIPS32 builds, does not produce MIPS-I cross-compilations anymore<sup>9</sup>. For example, even though we tried to run on the router our toolsets pre-compiled by Buildroot for MIPS target, certain binaries threw errors such as "*Illegal instruction*". This is the most tangible confirmation of a mismatch between the hardware CPU Instruction Set Architecture (ISA), and the ISA generated into the executable by the cross-compilation.

Addressing the "obsolete firmware environment" challenge is important for several reasons. Any system, whether offensive [51] or defensive such as *HALE-IoT*, will most likely face exactly similar challenge on a constantly and increasing basis. Some reasons for this are that devices become obsolete/EOL

faster and on a larger, and the technology and software development life-cycle is constantly accelerating. The above in turn implies several more things. First, to support many legacy IoT devices will require having an ever growing toolbox of cross-compilation environments. Such backward compatibility toolbox should provide as complete coverage as possible in terms of combinations for CPU ISA, OS/kernel, Application Binary Interfaces (ABI), runtime environments (including all different versions and inter-compatibility). Second, it will require human expertise and manual intervention to generate and maintain such cross-compilation environments, as well as to ensure that the target-specific builds of system such as *HALE-IoT* actually work without errors (e.g., "*Illegal instruction*").

#### F. Runtime Services Hardcoded to 0.0.0.0 :< port >

Sometimes network services (e.g., web-servers) have the port and the network interface binding hardcoded instead of being read from a configuration file (whether standard or proprietary one). This is problematic as it exposes the original built-in network server to potential attacks coming from attacker-accessible interfaces. In fact, in our dataset that expose web services, there were 32 firmwares that had either the interface and/or the port hardcoded directly into the binary. Ideally, with *HALE-IoT* (cf. Figure 1) the aim is to isolate (inherently) vulnerable services to 127.0.0.1 :< port >, and to expose via *HALE-IoT* only the hardened services.

One possible workaround to this challenge is to manually reverse-engineer and binary-patch the executable files of interest and ideally force them to bind to 127.0.0.1 :< port >. While this approach most likely will work in most cases, it cannot still scale similarly to the automated configuration change approach we presented above. Another possible workaround is to force-start a dummy TCP/UDP server on 0.0.0.0 :< port > before the built-in network service (e.g., web-server on port 80) has a chance to bind to it, and then observe how the original service behaves for re-binding (e.g., moves to another port, moves to another interface, fails to start altogether). Implementing and testing these adjustments is the scope of our future work.

#### G. From Self-signed HTTPS Certificates to Full CA Chains

One of the core aims of *HALE-IoT* is to generically harden IoT devices with proper HTTPS, including support for full chain certificates. For this, *HALE-IoT* approach uses the concept of HTTPS-proxying that creates a proper HTTPS service point that is being relayed to the built-in web server. Our current implementation choice is to use SSL-proxy, that provides latest and most secure TLS implementations, and supports full chain certificates. However, in order to simplify our experiments and for several practical reasons, we have used self-signed certificates generated by the SSL-proxy itself. Should we deploy and evaluate *HALE-IoT* on real-world internet facing IoT devices in the future, the following minimal steps would ensure an example implementation when using proper PKI full certificate chains:

- 1) Configure and connect the device to a public DNS sub-domain name under user's control (e.g.,

<sup>9</sup><https://github.com/buildroot/buildroot/blob/master/CHANGES>



- using DDNS services or otherwise), for example `device-X.fleet-Y.service-provider.com`.<sup>10</sup>
- 2) Have a full certificate chain issued by a trusted CA (e.g., Let's Encrypt), and covering `*.fleet-Y.service-provider.com` or `device-X.fleet-Y.service-provider.com` (depending on the desired granularity of “device identity management” versus “PKI/certificate/key control”).
  - 3) Use the corresponding full certificate chain and its private-key(s) to configure the SSL-proxy implant that goes into a corresponding device. This can be done before implanting *HALE-IoT*, or can be done while the hardened device is already running by using *HALE-IoT*'s SSH-based administrative interface.

The above is an example of the improvements needed to ensure a secure management of DNS, PKI, certificates, private-keys, and device identities.

Moreover, effective and efficient PKI implementations represent an ongoing area of research on its own [62], [63], especially when considering deployment of PKI for IoT [64], [65]. Therefore, we leave research, experimentation, and validation of full-blown PKI support for *HALE-IoT* as future work.

### H. Resource Constraints: Static Linking vs. Dynamic Loading

We present two approaches to deal with constraints in physical devices. On the one hand, static linking allows the toolsets within *HALE-IoT* to be self-contained, and not to depend on the existence of particular libraries within the target retrofitted firmwares. This makes the approach highly-scalable — cross-compile once, run everywhere. However, this approach considerably increases the size of the binaries included with the implant. This is problematic from a storage perspective, and from a memory perspective as there is essentially a possible duplication of library code loaded into RAM due to static linking. Flash storage and RAM memory are quite constrained and minimal in many devices. For instance, Asus RT-N12+ features a 32MB RAM chipset, where 28MB is allocated for userland applications, from which *less than 2.5MB was available for entire HALE-IoT*.

On the other hand, dynamic loading allows the toolsets to be built with minimal binary size and runtime RAM memory consumption, as there is no code duplication and the hardening toolsets can rely on the libraries already present on device's storage and RAM. However, this approach is highly non-scalable. For example, it means that the toolsets would have to be linked with dynamic loading to a myriad of library versions present in each different firmware version. Even if that could be automated somehow, it does not guarantee that the library exposes the correct and expected interface and functions (e.g., library is vendor-customized, missing headers).

This challenge is not easy to solve systematically. Our experience suggests that the best approach is to write highly-optimized toolsets design to fit into highly-constrained devices. If this is not feasible, then the newly developed hardening

toolsets, and especially the ones that are tailored specifically for IoT devices, should incorporate these design principles.

### I. Inherent Limitations of WAFs and xAFs

Like any piece of software, WAFs and xAFs or, in general, any implant that can be retrofitted into a system may be subject to limitations that can range from own implementation errors (e.g., WordPress WAF plugin recently vulnerable to SQLi itself<sup>11</sup>), or new vulnerabilities and zero-days that may arise in the future to outdated software configurations. Such fundamental limitations would also be inherited by *HALE-IoT*.

One limitation is that most WAFs can detect and prevent input-driven exploits, but very often they are unable to detect and prevent other attacks such as “stored XSS”. Another limitation is that WAFs are mostly rule-based, hence the presence or absence of specific rules may affect the successful of detection/prevention versus a failed one. Also, keeping such rules up-to-date is another factor that may affect or limit the effectiveness of any given WAF. We have designed *HALE-IoT* with an administrative interface in mind. This feature, if used often and correctly, may help overcome the limitations of outdated rule-sets and components. Though technically possible, we leave full automation of updating *HALE-IoT* rule-sets (and other components) as future work.

As a result of our work we have fixed several bugs in Raptor that have improved the overall reliability of the WAF. The discovery of the bugs and the development of their patches are a relatively modest contribution in itself. However, their discovery underpins the importance and the need of performing experiments such as ours, hence exposing well-known and widely-used software to even more scenarios.

**Bugs in Raptor WAF – HTTPS.** During the course of the experiments, we detected certain bugs in the way the WAF should work. First, the communication lasted longer than expected even when all the response had been received, causing, for example, the browser to appear to still be loading the web page. After a little inspection in the source code we observed that the socket descriptor was not closing, which caused the connection to remain established. Second, we encountered strange behavior when the WAF was running alongside the SSL-proxy. In this case, when we made POST type HTTP requests that included some payload, Raptor did not detect them correctly. When only the WAF was running instead, it worked as expected with the same request. After a closer inspection, we found out that the WAF checks whether it is a web request or not, and analyzes this in case it is. Most web clients (e.g., curl or web browsers) send the headers and the data in the same packet (unless the data is very large that it will be divided into different TCP packets). However, the HTTP libraries of the Go programming language, split the request — first send the headers, and then the data itself. Hence, Raptor WAF fails to analyze the data from following GET/POST request(s). We patched Raptor's code to check the data size of the headers, and then reassembling the packets before analyzing and proxying the traffic to the web server.

<sup>10</sup>A full first-level DNS domain name would also work for a single device, but is sub-optimal and hardly practical for managing larger fleets of devices.

<sup>11</sup><https://portswigger.net/daily-swig/wordpress-security-plugin-hide-my-wp-addresses-sql-injection-deactivation-flaws>

**Bugs in Raptor WAF – Memory Leaks.** We found several memory-related problems in Raptor WAF. First, we encountered a *use-after-return* error [66]. This error occurs when a function returns the memory address of a local variable, which is “destroyed” when the function terminates. Therefore, the returned pointer references to an area of the stack that could be used for another function, and could cause unwanted behavior or exploitation of the program [67]. Finally, we found several cases where dynamically allocated memory areas are not properly released, which caused memory leaks. Not freeing up memory causes the program to eventually store more memory than it needs, which is a major issue with memory-constrained devices (see also Section VII-H). This can be an overall limiting factor to the usability of the retrofitting implants, and can also lead to general instability and crashes of the systems where Raptor WAF is installed, i.e., can affect high-end servers and not just constrained IoT devices. Lastly, there is inherent risk of the uninitialized memory created by the dynamic allocation algorithm. This is not a security bug per-se (rather a feature of many programming languages), but certain functions such as `malloc` return a pointer to a block of memory that has uninitialized values and can be potentially exploited [68], [69]. However, the shortcoming of having uninitialized memory areas can be effectively remedied by making use of the `calloc` function which fills the dynamically allocated memory block with zeros as deterministic initial value, at the expense of minor performance overheads.

## VIII. RELATED WORK

Hardening legacy IoT devices has been a subject of few research papers over the years [14], [20], [22], [70]–[73]. Related works follow different strategies, we group them in three categories: i) embedding defensive software or retrofitting security measures, ii) securing firmware from malicious modifications, and iii) securing access control and communications.

### A. Retrofitting, Patching, and Hardening for Security

Enhancing the security of a single IoT device is a defensive strategy that works best when the devices are not part of a large centralized network of IoT devices. Cui and Stolfo [20] introduce the notion of Symbiotic Embedded Machine (SEM), a software design to embed defensive software into an existing installation. The authors embedded an intrusion detection system and show how these strategies can lead to the detection of stealthy malware (i.e., a rootkit) into a Cisco router. Choi et al. [14] followed similar approach in their research. They developed a scheme to deploy security features in poorly-secured IoT devices through the deployment of a monitoring Web service that manages multiple IoT devices in a network. Recently, Maroof et al. [74] presented iRECOVer, a holistic solution for the security management IoT devices. It aims to replace “vulnerable modules” with “secure modules” and offering “secure channels” for communicating devices, without specifically addressing backward compatibility and equivalent intended functionality of secured modules. The authors demonstrated iRECOVer only on a single Raspberry-Pi 4 Model B device

running customized open-source Linux distribution running. The authors were unable to demonstrate iRECOVer on real-world IoT devices, as they acknowledge that “*programming a commodity IoT device is difficult*”. *HALE-IoT* is somewhat fundamentally different as it is demonstrated to work on a large number of heterogeneous and original commodity IoT devices/firmware, and does not replace original modules but rather wraps them with added-secured layers. These works came to show that retrofitting security measures is a process agnostic to the platform (hardware and software), and it does not need to attune to any executable format. Similarly to approaches based on SEM, *HALE-IoT* is installed alongside the original operating system and injects protecting payloads into the target. However, prior work propose hardening solutions that are tailored to specific attacks and are limited by the scale and lack of automation. *HALE-IoT* is designed as a generic method to deploy universal and hardening solutions with proven effectiveness, while at the same time minimizing intrusion and re-configurations to the original firmware.

Not every hardening tool is universal as some target to secure a more specific section of IoT devices. For example, Christensen et al. [70] introduced DECAF, their Unified Extensible Firmware Interface (UEFI) firmware code pruning system to reduce redundant and possibly vulnerable code from firmware while increasing system performance and security. These firmware commonly exist in motherboards. The DECAF platform does this “debloating” by performing “dynamic iterative surgery” and utilizing existing knowledge of the firmware via removing possible known issues. The authors declare that in some cases, DECAF has reduced the UEFI firmware code by over 70%; thus, notably reducing the attack surface of the firmware. The authors claim that DECAF could potentially be extended to prune any type of firmware. Similarly, Cui et al. [51] propose ABSR as an early conceptual ideal for firmware code-debloating technique achieved via binary-patching and binary-rewriting. Recently, Zhang et al. [75] presented *μTrimmer*, a system to identify and remove unused basic blocks from binary code of shared libraries and tools. The authors implemented *μTrimmer* for MIPS architecture (a very common one for IoT devices), and tested its effectiveness on SPEC CPU2017 benchmarks, popular firmware applications (e.g., OpenSSL), and a single real-world wireless router firmware image. *μTrimmer* demonstrated that static library debloating on stripped binaries challenge, while being enormous, is not insurmountable for MIPS-based firmware – their system produced functional programs while reducing unnecessary exposed code surface and remarkably eliminating various reusable code gadgets. However, debloating itself is ineffective at hardening core services in the firmware (e.g., fragments of the firmware that cannot, or should not, be pruned). Additionally, debloating in principle is a high-risk technique as it may prune code segments that are instrumental for normal and intended operation of the system/device as a whole. Our system avoids debloating altogether and hardens the potentially vulnerable services with securing wrappers that bring proven effectiveness (e.g., Raptor WAF) and security guarantees (e.g., SSL-proxy).

Standalone IoT devices often interface with the user via

built-in web servers due to its wide and cost-effective adoption. However, web services often introduce vulnerabilities to the system. Gourdin et al. [76] tackled this issue by developing WebDroid, an IoT focused Android OS web interface developing framework with security as a priority. WebDroid enables developers to easily create more secure web interfaces for their Android-based IoT devices. The framework takes into account many important security issues such as bad authentication practices, cross-site scripting (XSS), and cross-site request forgery (CSRF). These frameworks are an interesting first step towards securing devices for vendors that lack means to produce secure environments [77]. However, these type of frameworks are meant to be integrated into the source-code and development life-cycle, and can not be easily adopted to secure firmware already deployed. Our work, on the other hand, practically demonstrates a systematic approach to integrating defensive measures post-deployment and without access to the source-code.

### B. Malicious Firmware Modifications

Other related works perform firmware modification to attack devices [73], such as malware targeting Universal Serial Bus (USB) devices [78] or printers [51], as well as attacks to critical infrastructures [79] (including smart grids [80]). These works are certainly a strong testament that firmware modifications have important real-world implications. However, modifying the firmware to embed defensive and protective mechanisms (as we do with *HALE-IoT*) requires an entire methodological consideration and evaluation to both preserve the correct functioning of the device (§ V-E) and to assess the real effectiveness of the multi-dimensional defenses (§ V-D).

### C. Authentication and Encryption of IoT Communications

Perhaps weak authentication, lack of encryption and vulnerable web services are altogether one of the largest attack surfaces in IoT devices to date [7], [26]–[28], [81]–[83]. There is a significant number of previous work that:

- 1) measures and points out crypto-security deficiencies in the IoT realm as a whole [7],
- 2) identifies the use of weak cryptography in constrained devices [27], [28], or
- 3) discovers weaknesses in the way TLS or PKI is deployed over the IoT [26], [82].

However, none of the existing works in the literature manage to effectively harden these services due to its intrinsic complexity — “*HTTPS deployment process is far too complex even for people with proficient knowledge in the field*” [84], not to mention when such deployment is rooted into an obscure component such as the firmware of a legacy IoT device. It is also well known that the web is “*large and complicated enough to make even conceptually simple security upgrades challenging to deploy in practice*” [85]. The web of IoTs of all networks is perhaps one of the *hardest to harden*. Still, our work presents a practical, sound, and actionable contribution towards addressing these non-trivial challenges.

### D. Over-The-Air (OTA) Firmware and Software Updates

Kolehmainen [86] performed a survey of secure firmware updates for IoT. The author concluded that there are virtually as many firmware and software update procedures as there are manufacturers, and proposed a common four element update model – packing, delivery, authentication, and attestation. Bauwens et al. [87] summarized and outlined key-principles of OTA for IoT devices and deployments.

Regarding (secure) OTA and Firmware OTA (FOTA) implementations, the automotive industry is perhaps the forerunner and trend-setter within the research literature. Idrees et al. [88] showcased a model for manufacturers, workshops, and vehicles to establish secure end-to-end link using a trusted platform model and secure communication. The model can be used to secure FOTA updates. Chowdhury et al. [89] proposed an ISO 26262 and SAE J3061 utilizing assurance case template for OTA updates. If used properly, the template is a valuable tool for manufacturers to root out security issues in their automotive OTA implementation in the development phase.

In 2020, Halder et al. [10] conducted a survey in regards to OTA updates of network connected vehicles. The identified some challenges that the industry is yet to fully solve. For example, the software distribution needs to protect privacy as well as be secure. Latency of the software installation can also be an issue especially to autonomous vehicles. Furthermore, key management is generally based on the trust of pre-installed keys, considering the lifetime of an automobile, key refresh may be in order and propose issues.

However, our present work have a different and complementary impact, meaning that *HALE-IoT* does not propose to solve any challenges faced by (secure) OTA/FOTA software updates. In fact, *HALE-IoT* itself could be delivered/deployed by any OTA/FOTA system that is running (or supported) by the particular device(s). We leave the exploration of integrating *HALE-IoT* into OTA/FOTA workflows as promising future work.

## IX. CONCLUSION

The Internet and private networks are littered with millions of vulnerable IoT devices. A large number of these devices are effectively *abandoned* by manufactures, who do not issue patches to fix known issues. This prevents users and network administrators from keeping their devices up to date, and thus poses an endemic risk to the security of the Internet, as well as of the enterprise and private/home networks. Hardening IoT devices allows to reduce the attack surface, emerging as a promising countermeasure. However, prior work is limited by scope of their contribution, and clearly fail to deal with the heterogeneity and the many technological constraints of both modern and legacy IoT devices.

In this paper, we presented a systematic methodology designed to retrofit sophisticated state-of-the-art defensive mechanisms into IoT firmware, with particular focus on legacy and obsolete firmware. We tested our framework with a wide-range of firmware images from different vendors and heterogeneous architectures, totaling 395 emulated firmware and four physical devices. Our results demonstrated that *HALE-IoT* successfully retrofits defensive implants in a scalable and

safe manner (i.e., without breaking the firmware). We also evaluated the performance of our approach under a battery of one hundred attacks, showing it is feasible to deploy *HALE-IoT* in the wild.

We discussed our findings and identified a number of limitations that came to show the challenges behind dealing with the idiosyncrasies of hardening IoT devices. Our discussion elicited also a number of future promising directions. First, an interesting avenue of research is to explore the use of defense-in-depth strategies as a mechanism to harden IoT devices. This introduces non-negligible trade-offs between the complexity of the method (e.g., iptables, Snort, fail2ban) and the overall gain. Second, we identify the need to automate the cross-compilation of the implants to more CPU architectures (e.g., RISC-V, or Xtensa) and to a more diverse set of obsolete environments (e.g., MIPS-I), while minimizing the overall footprints at build and runtime (e.g., storage, RAM, CPU). Finally, we would like to encourage researcher and industry practitioners having access to large sets of physical devices to enlarge the evaluation and their support in *HALE-IoT*.

#### ACKNOWLEDGMENTS

The authors would like to thank Andrei Costin for hosting Javier Carrillo at the University of Jyväskylä and “call for stays at universities and research centers abroad for the year 2021” from University of Castilla-La Mancha for supporting this research visit. Javier Carrillo is also supported by the MINECO and European Commission (FEDER funds) under project RTI2018-098156-B-C52, by the JCCM under the project SB-PLY/17/180501/000353, and by the Spanish Education, Culture and Sports Ministry under grant FPU 17/03105. Hannu Turtiainen also thanks the Finnish Cultural Foundation / Suomen Kulttuurirahasto (<https://skr.fi/en>) for supporting his Ph.D. dissertation work and research (under grant decision no. 00221059) and the Faculty of Information Technology of the University of Jyväskylä (JYU), in particular, Prof. Timo Hämäläinen, for partly supporting and supervising his Ph.D. work at JYU in 2021–2022. Guillermo Suarez-Tangil was funded by the “Ramon y Cajal” Fellowship RYC-2020-029401-I. The authors thank Jami Laamanen for his contributions during early-stage of the experiments [90]. The authors also acknowledge the use of royalty-free icons courtesy of [www.flaticon.com](http://www.flaticon.com) (icons by: Cuputo, Good Ware, rukanicon, and Prosymbols).

#### REFERENCES

- [1] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, “A large-scale analysis of the security of embedded firmwares,” in *23rd USENIX Security Symposium*, 2014.
- [2] A. Costin, A. Zarras, and A. Francillon, “Automated dynamic firmware analysis at scale: a case study on embedded web interfaces,” in *11th ACM on Asia Conference on Computer and Communications Security*, 2016.
- [3] D. Chen, M. Egele, M. Woo, and D. Brumley, “Towards Automated Dynamic Analysis for Linux-based Embedded Firmware,” in *NDSS*, vol. 25, 2016.
- [4] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztain, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, “Understanding the mirai botnet,” in *26th {USENIX} Security Symposium*, 2017.
- [5] A. Cui and S. J. Stolfo, “A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan,” in *26th Annual Computer Security Applications Conference (ACSAC)*, 2010.
- [6] Carna Botnet, “Internet Census 2012: Port scanning/0 using insecure embedded devices,” *SourceForge, White Paper*, 2012.
- [7] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, “Mining your Ps and Qs: Detection of widespread weak keys in network devices,” in *21st {USENIX} Security Symposium*, 2012.
- [8] A. Mirian, Z. Ma, D. Adrian, M. Tischler, T. Chuenchujit, T. Yardley, R. Berthier, J. Mason, Z. Durumeric, J. A. Halderman *et al.*, “An Internet-wide view of ICS devices,” in *14th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2016.
- [9] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, “All things considered: an analysis of IoT devices on home networks,” in *28th {USENIX} Security Symposium*, 2019.
- [10] S. Halder, A. Ghosal, and M. Conti, “Secure over-the-air software updates in connected vehicles: A survey,” *Computer Networks*, vol. 178, 2020.
- [11] O. Alrawi, C. Lever, K. Valakuzhy, R. Court, K. Snow, F. Monrose, and M. Antonakakis, “The Circle Of Life: A Large-Scale Study of The IoT Malware Lifecycle,” in *30th USENIX Security Symposium*, 2021.
- [12] R. Millman, “IoT devices are more vulnerable than ever,” 2021, <https://www.itpro.co.uk/network-internet/internet-of-things-iot/360850/iot-devices-are-more-vulnerable-than-ever>.
- [13] S. Weston, “83 million IoT devices at risk of hacking,” 2021, <https://www.itpro.co.uk/network-internet/internet-of-things-iot/360612/83-million-iot-devices-at-risk-of-hacking>.
- [14] S.-K. Choi, C.-H. Yang, and J. Kwak, “System hardening and security monitoring for IoT devices to mitigate IoT security vulnerabilities and threats,” *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 12, 2018.
- [15] A. Nguyen-Tuong, S. Guarnieri, D. Greene, J. Shirley, and D. Evans, “Automatically hardening web applications using precise tainting,” in *IFIP International Information Security Conference*. Springer, 2005.
- [16] T. Fraser, L. Badger, and M. Feldman, “Hardening COTS software with generic software wrappers,” in *IEEE Symposium on Security and Privacy*. IEEE, 1999.
- [17] S. Kubler, K. Främling, and A. Buda, “A standardized approach to deal with firewall and mobility policies in the IoT,” *Pervasive and Mobile Computing*, vol. 20, 2015.
- [18] N. Gupta, V. Naik, and S. Sengupta, “A firewall for internet of things,” in *9th International Conference on Communication Systems and Networks (COMSNETS)*. IEEE, 2017.
- [19] B. P. Sindhuri and M. K. Rao, “IoT security through web application firewall,” *Int. J. Eng. Technol.*, vol. 7, 2018.
- [20] A. Cui and S. J. Stolfo, “Defending embedded systems with software symbiotes,” in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2011.
- [21] C. Ye, P. P. Indra, and D. Aspinall, “Retrofitting security and privacy measures to smart home devices,” in *6th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. IEEE, 2019.
- [22] C. Frank, C. Nance, S. Jarocki, and W. E. Pauli, “Protecting IoT from Mirai botnets; IoT device hardening,” *Journal of Information Systems Applied Research*, vol. 11, 2018.
- [23] D. Stuttard and M. Pinto, *The web application hacker’s handbook: Finding and exploiting security flaws*. John Wiley & Sons, 2011.
- [24] H. Bojinov, E. Bursztain, E. Lovett, and D. Boneh, “Embedded management interfaces: Emerging massive insecurity,” *BlackHat USA*, 2009.
- [25] M. Dahlmans, J. Lohmöller, J. Pennekamp, J. Bodenhausen, K. Wehrle, and M. Henze, “Missed Opportunities: Measuring the Untapped TLS Support in the Industrial Internet of Things,” in *17th ACM on Asia Conference on Computer and Communications Security*, 2022.
- [26] T. Chung, Y. Liu, D. Hoffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, “Measuring and applying invalid ssl certificates: The silent majority,” in *Internet Measurement Conference*, 2016.
- [27] N. Samarasinghe and M. Mannan, “Short paper: TLS ecosystems in networked devices vs. web servers,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2017.
- [28] —, “Another look at TLS ecosystems in networked devices vs. Web servers,” *Computers & Security*, vol. 80, 2019.
- [29] CoolerVoid, “raptor\_waf,” 2021. [Online]. Available: [https://github.com/CoolerVoid/raptor\\_waf](https://github.com/CoolerVoid/raptor_waf)
- [30] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztain, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman,

## Chapter 7. HALE-IoT: HARDening Legacy Internet-of-Things devices via retrofitting defensive firmware modifications and implants

18

- N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>
- [31] G. Luptak and D. Palotay, "IoT Botnet Report 2021: Malware and Vulnerabilities Targeted," 2021, <https://cujo.com/iot-botnet-report-2021-malware-and-vulnerabilities-targeted/>.
- [32] A. Costin and J. Zaddach, "IoT malware: Comprehensive survey, analysis framework and case studies," 2018.
- [33] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding linux malware," in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 161–175.
- [34] E. Cozzi, P.-A. Vervier, M. Dell'Amico, Y. Shen, L. Bilge, and D. Balzarotti, "The tangled genealogy of IoT malware," in *Annual Computer Security Applications Conference*, 2020, pp. 1–16.
- [35] B. Zhao, S. Ji, W.-H. Lee, C. Lin, H. Weng, J. Wu, P. Zhou, L. Fang, and R. Beyah, "A large-scale empirical study on the vulnerability of deployed iot devices," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2020.
- [36] O. Alrawi, C. Lever, K. Valakuzhy, R. Court, K. Snow, F. Monrose, and M. Antonakakis, "The circle of life: A Large-Scale study of the IoT malware lifecycle," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3505–3522. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity21/presentation/alrawi-circle>
- [37] T. Gilb and S. Finzi, *Principles of software engineering management*. Addison-Wesley Reading, MA, 1988, vol. 11.
- [38] S. Kumar, "suyashkumar/ssl-proxy," 2021. [Online]. Available: <https://github.com/suyashkumar/ssl-proxy>
- [39] "Home - Lighttpd - fly light." [Online]. Available: <https://www.lighttpd.net/>
- [40] "Let's Encrypt." [Online]. Available: <https://letsencrypt.org/>
- [41] ReFirmLabs, "Binwalk," <https://github.com/ReFirmLabs/binwalk>.
- [42] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna, "Firmalce – Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware," in *NDSS*, vol. 24, 2015.
- [43] "Accelerating deployment of Arm-based solutions." [Online]. Available: <https://www.linaro.org/>
- [44] "Linux Toolchain – MIPS." [Online]. Available: <https://www.mips.com/develop/tools/compilers/linux-toolchain/>
- [45] "Buildroot - Making Embedded Linux Easy." [Online]. Available: <https://buildroot.org/>
- [46] "musl libc." [Online]. Available: <https://musl.libc.org/>
- [47] L. Bass, I. Weber, and L. Zhu, *DevOps: A software architect's perspective*. Addison-Wesley Professional, 2015.
- [48] C. Heistand, J. Thomas, N. Tzeng, A. R. Badger, L. M. Rodriguez, A. Dalton, J. Pai, A. Bodzas, and D. Thompson, "DevOps for Spacecraft Flight Software," in *IEEE Aerospace Conference*. IEEE, 2019.
- [49] "payloadbox/sql-injection-payload-list," 2021. [Online]. Available: <https://github.com/payloadbox/sql-injection-payload-list>
- [50] "payloadbox/xss-payload-list," 2021. [Online]. Available: <https://github.com/payloadbox/xss-payload-list>
- [51] A. Cui, M. Costello, and S. Stolfo, "When firmware modifications attack: A case study of embedded exploitation," *NDSS*, vol. 22, 2013.
- [52] A. Hemel, "Using ELF symbols extracted from dynamically linked ELF binaries for fingerprinting," 2021.
- [53] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," in *ACM SIGSAC Conference on Computer and Communications Security*, 2017.
- [54] J. C. (db90h) and C. Heffner, "Firmware Modification Kit."
- [55] D. Giese, "How-to modify ARM Cortex-M based firmware: A step-by-step approach for Xiaomi IoT Devices," [https://dontvacuum.me/talks/DEFCON26-IoT-Village/DEFCON26-IoT-Village\\_How\\_to\\_Modify\\_Cortex\\_M\\_Firmware-Xiaomi.pdf](https://dontvacuum.me/talks/DEFCON26-IoT-Village/DEFCON26-IoT-Village_How_to_Modify_Cortex_M_Firmware-Xiaomi.pdf), 2018.
- [56] D. Goodin, "A vigilante is putting a huge amount of work into infecting IoT devices," <https://arstechnica.com/information-technology/2017/04/a-vigilante-is-putting-huge-amount-of-work-into-infecting-iot-devices/>, 2017.
- [57] O. Shwartz, Y. Mathov, M. Bohadana, Y. Elovici, and Y. Oren, "Opening Pandora's box: effective techniques for reverse engineering IoT devices," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2017.
- [58] J. Zaddach, L. Bruno, A. Francillon, D. Balzarotti et al., "Avatar: A framework to support dynamic security analysis of embedded systems' firmwares," in *NDSS*, vol. 23, 2014.
- [59] S. Vasile, D. Oswald, and T. Chothia, "Breaking all the things – A systematic survey of firmware extraction techniques for IoT devices," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2018.
- [60] A. Hemel, "binaryanalysis-ng," <https://github.com/armijnhemel/binaryanalysis-ng>.
- [61] I. Skochinsky, "Intro to embedded reverse engineering for pc reversers," in *REcon conference, Montreal, Canada*, 2010.
- [62] P. Gutmann, "PKI: it's not dead, just resting," *Computer*, vol. 35, no. 8, pp. 41–49, 2002.
- [63] C. Adams and S. Lloyd, *Understanding PKI: concepts, standards, and deployment considerations*. Addison-Wesley Professional, 2003.
- [64] Z.-K. Zhang, M. C. Y. Cho, and S. Shieh, "Emerging security threats and countermeasures in IoT," in *Proceedings of the 10th ACM symposium on information, computer and communications security*, 2015, pp. 1–6.
- [65] D. Diaz-Sanchez, A. Marín-Lopez, F. A. Mendoza, P. A. Cabarcos, and R. S. Sherratt, "TLS/PKI challenges and certificate pinning techniques for IoT and M2M secure communications," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3502–3531, 2019.
- [66] A. Samsonov and K. Serebryany, "New features in addresssanitizer," 2013.
- [67] P. Akrividis, C. Cadar, C. Raiciu, M. Costa, and M. Castro, "Preventing memory error exploits with WIT," in *2008 IEEE Symposium on Security and Privacy*. IEEE, 2008.
- [68] S. Bhatkar, D. C. DuVarney, and R. Sekar, "Efficient Techniques for Comprehensive Protection from Memory Error Exploits," in *USENIX Security Symposium*, vol. 10, 2005, pp. 1 251 398–1 251 415.
- [69] A. Milburn, H. Bos, and C. Giuffrida, "Safelnit: Comprehensive and Practical Mitigation of Uninitialized Read Vulnerabilities," in *NDSS*, vol. 17, 2017, pp. 1–15.
- [70] J. Christensen, I. M. Anghel, R. Taglang, M. Chiroiu, and R. Sion, "{DECAF}: Automatic, Adaptive De-bloating and Hardening of {COTS} Firmware," in *29th {USENIX} Security Symposium*, 2020.
- [71] Z. Grimmer, J. Staggs, and S. Shenoi, "Retrofitting mobile devices for capturing memory-resident malware based on system side-effects," in *IFIP International Conference on Digital Forensics*. Springer, 2019.
- [72] C. Segarra, R. Delgado-Gonzalo, and V. Schiavoni, "MQT-TZ: Hardening IoT Brokers Using ARM TrustZone: (Practical Experience Report)," in *International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2020.
- [73] J. Zaddach, A. Kurmus, D. Balzarotti, E.-O. Blass, A. Francillon, T. Goodspeed, M. Gupta, and I. Koltidas, "Implementation and implications of a stealth hard-drive backdoor," in *29th Annual Computer Security Applications Conference (ACSAC)*, 2013.
- [74] U. Maroof, A. Shaghagh, R. Michelin, and S. Jha, "iRECOVer: Patch your IoT on-the-fly," *Future Generation Computer Systems*, 2022.
- [75] H. Zhang, M. Ren, Y. Lei, and J. Ming, "One size does not fit all: security hardening of mips embedded systems via static binary debloating for shared libraries," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022.
- [76] B. Gourdin, C. Soman, H. Bojinov, and E. Bursztin, "Toward secure embedded web interfaces," in *20th USENIX Security Symposium*, 2011.
- [77] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *IEEE Symposium on Security and Privacy*. IEEE, 2019.
- [78] Cybersecurity and I. S. Agency, "Security Tip (ST08-001) – Using Caution with USB Drives," 2019. [Online]. Available: <https://us-cert.cisa.gov/ncas/tips/ST08-001>
- [79] Z. Basnight, J. Butts, J. Lopez Jr, and T. Dube, "Firmware modification attacks on programmable logic controllers," *International Journal of Critical Infrastructure Protection*, vol. 6, 2013.
- [80] C. Konstantinou and M. Maniatakos, "Impact of firmware modification attacks on power systems field devices," in *IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2015.
- [81] L. Valenta, S. Cohnay, A. Liao, J. Fried, S. Bodduluri, and N. Heninger, "Factoring as a service," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016.
- [82] M. T. Paracha, D. J. Dubois, N. Vallina-Rodriguez, D. Choffnes et al., "IoTLS: Understanding TLS Usage in Consumer IoT Devices," in *Internet Measurement Conference*, 2021.
- [83] R. T. Tiburski, L. A. Amaral, E. de Matos, D. F. de Azevedo, and F. Hessel, "Evaluating the use of TLS and DTLS protocols in IoT middleware systems applied to E-health," in *14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2017.

- [84] K. Krombholz, W. Mayer, M. Schmiedecker, and E. Weippl, “‘’ i have no idea what i’m doing’-on the usability of deploying {HTTPS},” in *26th {USENIX} Security Symposium*, 2017.
- [85] M. Kranch and J. Bonneau, “Upgrading HTTPS in mid-air,” in *NDSS*, vol. 24, 2015.
- [86] A. Kolehmainen, “Secure firmware updates for IoT: a survey,” in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 112–117.
- [87] J. Bauwens, P. Ruckebusch, S. Giannoulis, I. Moerman, and E. De Poorter, “Over-the-air software updates in the internet of things: An overview of key principles,” *IEEE Communications Magazine*, vol. 58, no. 2, pp. 35–41, 2020.
- [88] M. S. Idrees, H. Schweppe, Y. Roudier, M. Wolf, D. Scheuermann, and O. Henniger, “Secure automotive on-board protocols: A case of over-the-air firmware updates,” in *International Workshop on Communication Technologies for Vehicles*. Springer, 2011, pp. 224–238.
- [89] T. Chowdhury, E. Lesiuta, K. Rikley, C.-W. Lin, E. Kang, B. Kim, S. Shiraishi, M. Lawford, and A. Wassyn, “Safe and secure automotive over-the-air updates,” in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2018, pp. 172–187.
- [90] J. Laamanen, “Hardening legacy IoT-devices by retrofitting security measures,” *MSc thesis*, 2019.

## APPENDIX

### A. Supporting Materials – Screenshots

#### 1) Screenshots for Evaluation on IPTV camera (emulation):

Figure 6 shows the TRENDnet TV-IP121WN IP camera web interface running with *HALE-IoT* via firmware emulation.

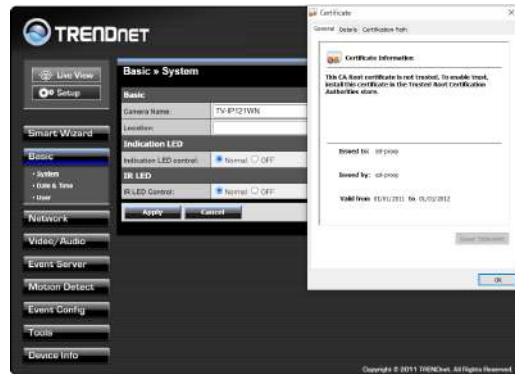


Fig. 6. Screenshot depicting emulated TRENDnet TV-IP121WN IPcam's web-interface along with the generated HTTPS certificate as part of the SSL/TLS hardening by *HALE-IoT* (emulation).

#### 2) Screenshots for Evaluation on CVE-2016-1555 (emulation):

Figure 7 shows the successful detection of the attempted exploitation of CVE-2016-1555 (XSS and Command Injection) on an emulated firmware that is hardened by *HALE-IoT*.

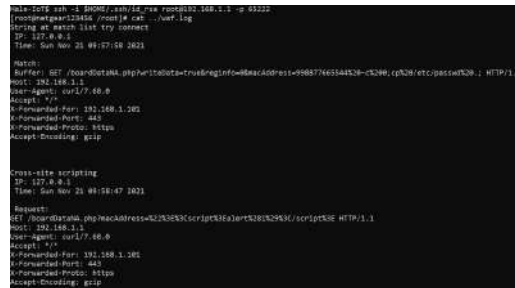


Fig. 7. Evaluation on CVE-2016-1555 (emulation): Screenshot depicting successful detection and prevention of both XSS and Command Injection attacks attempting to exploit CVE-2016-1555 on an emulated firmware hardened with *HALE-IoT*.

#### 3) Screenshots for Evaluation on RPi3 with OpenWrt (device):

Figure 8 and Figure 9 show how *HALE-IoT* works on a RaspberryPi 3 running OpenWrt (Section VI-B). Figure 8 shows the successful exploitation of an XSS in the LuCI web interface without *HALE-IoT*, while Figure 9 shows that the attack is detected and prevented when the device is hardened by *HALE-IoT*.

#### 4) Screenshots for Evaluation on several devices (device):

Figure 10, 12 and 11 show the successful installation of *HALE-IoT* on the physical devices Asus RT-N12+ B1 (Section VI-C),



Fig. 8. Evaluation on RPi3 with OpenWrt (device): Screenshot depicting XSS in OpenWrt's LuCI web interface running on RPi3 without *HALE-IoT*.

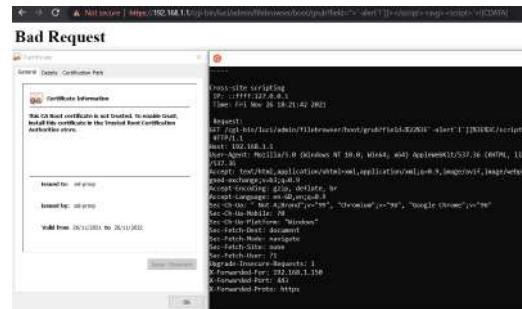


Fig. 9. Evaluation on RPi3 with OpenWrt (device): Screenshot depicting successful prevention of XSS and addition of HTTPS after RPi3 running the same originally vulnerable OpenWrt was hardened with *HALE-IoT*.

Netgear R6220 (Section VI-D), and Linksys EA4500 (Section VI-E), respectively.

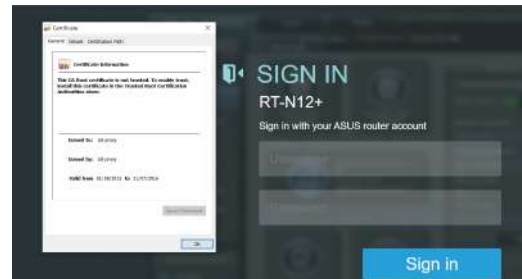


Fig. 10. Evaluation on Asus RT-N12+ (device): Screenshot depicting *HALE-IoT* implant successfully running on Asus RT-N12+ B1 device.

#### 5) Screenshots for Evaluation on Linksys wrt54gs Hard-coded Binary (emulation):

Figure 13 shows the Ghidra decompiled code of the BusyBox httpd applet showing that it accepts an option to change the IP address to bind the web server. On the other hand, Figure 14 shows that this option is not available in the help menu of the httpd applet. Finally, Figure 15 shows that through this option it is possible to isolate the web server when it is working with *HALE-IoT* (Section VI-A).



Fig. 11. Screenshot depicting Netgear R6220's web-interface along with the generated HTTPS certificate as part of the SSL/TLS hardening by *HALE-IoT*.



Fig. 14. Linksys wrt54gs Hardcoded Binary (emulation): Screenshot depicting that the builtin *httpd* server's help menu does not normally show the hidden *“-l”* option.

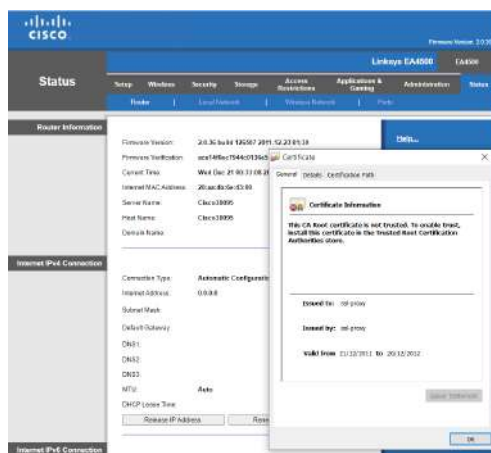


Fig. 12. Screenshot depicting Linksys EA4500's web-interface along with the generated HTTPS certificate as part of the SSL/TLS hardening by *HALE-IoT*.



Fig. 15. Linksys wrt54gs Hardcoded Binary (emulation): Screenshot depicting that builtin *httpd* server did a successful re-bind to 127.0.0.1 by forcing this via the hidden “-l” option.

```

[DAT_1000018 + 0x2030] = "Web Server Authentication";
[DAT_1000018 + 0x2032] = 0x0;
uvar2 = kb_getopt_u16(params, param_2, &id, &triple_id);
0x2000018 + 0x2014, &local_40,
&local_44, &local_48, &DAT_1000018 + 0x2008, &local_3c, &local_36, &local_34;

local_5 = DAT_1000018;
if (0x0 & 2) != 0 {
  param_3 = R_A_048724(&local_40, 1);
  JB_00459040;
}
if ((local_2 & 0x00) != 0) {
  *(&DAT_1000018 + 0x2030) = &local_34;
}
[DAT_1000018 + 0x2050] = uvar2 & 0x10;
uvar3 = &id(&local_40);
if (uvar3 != 0) {
  kb_perror_msg_and_exit("can't chdir to %s", &local_44);
}
memset(&local_100_0, 0, 0x10);
local_100_0_2_2 = 2;
uvar1 = &id(&id) * (&DAT_1000018 + 0x2030), &local_104;
if (uvar3 == 1) {
  if (*(&DAT_1000018 + 0x2030) == 0x0) {
    local_10010 = 0;
  }
  else {
    param_4 = gethostname(*(&DAT_1000018 + 0x2030));
    local_104(0) = param_4, &uvar1, &id, &uvar3, &uvar3;
  }
}
local_100_2_2_2 = *(&DAT_1000018 + 0x202c) >> 8 | (*(&DAT_1000018 + 0x202c) & 0xFF) << 8;
uvar2 = socket(2, 0, 0);
if (uvar2 < 0) {
  param_2 = "create socket";
}
else {
  local_30 = 1;
  netpoll(&local_2_2_2, &uvar1, 4, &local_30, 0);
  uvar3 = bind(uvar2, &local_100_0, 0x10);
}

```

Fig. 13. Linksys wrt54gs Hardcoded Binary (emulation): Screenshot depicting the presence of a hidden “-l” option used for binding network interface.



# CHAPTER 8

## Conclusions and Future Work

The purpose of this chapter is to conclude the doctoral thesis by summarizing the conclusions drawn from this work. In addition, it also discusses some lines of research that follow on from this proposal.

### 8.1 Conclusions

The rapid and growing development of the IoT ecosystem brings us services and utilities that a few years ago seemed unthinkable. However, this has led to a scenario in which everything is connected, prompting security and privacy concerns. In recent years, the number of attacks on this type of device has increased and will continue to do so in the years to come. Therefore, having solutions to understand the actions carried out by attackers, as well as tools to analyze and detect attacks, is an important aspect of cybersecurity.

In view of the above, the main conclusion of this doctoral thesis is that there is a need to **model, analyze and detect attacks in the IoT ecosystem**, which has been the main objective of this doctoral thesis. To achieve this goal, the actions carried out by the attackers have been measured, and tools have been designed and implemented to analyze and correlate malicious software. The last step was to design and implement a methodology for adding additional security measures to IoT devices.

In order to fulfill this objective, several goals were established to address the proposal in the correct way. The first goal was centered on **the analysis of the proposals from the scientific community regarding the classification and detection of threats in IoT environments**, and from this the following conclusions can be drawn:

- Technology is constantly evolving and it is crucial to understand the latest proposals in order to keep at the cutting edge in the research process.
- The IoT ecosystem is very heterogeneous, being made up of different types of devices with highly varied hardware and software architectures, and interconnected to each

## 8.1. Conclusions

---

other through different communication protocols. All this has led to the appearance of inherent vulnerabilities in this ecosystem and a perfect scenario for cybercriminals, who take advantage of these vulnerabilities to exploit them in the wild and get access to devices.

- Malware targeting Linux/IoT-based systems is continually growing and, unlike that of other ecosystems such as Windows and Android, there are not many tools available to perform dynamic analysis, mainly because IoT malware is compiled for different hardware architectures (e.g., PowerPC, ARM, MIPS).

Regarding the second goal, which focused on **the analysis of attack trends in the IoT ecosystem**, this allowed the study of two honeypots. The first was focused on Telnet and SSH services and the actions that attackers take once they are in control. From this analysis we can conclude that brute force attacks on these services remain one of the preferred methods for taking control of, and infecting, devices. Most of the sessions established were carried out automatically, and they can be related by the commands they used once they accessed the device. These types of honeypots are an effective way of collecting new variants of malware that attack IoT devices and valuable information about the actions they take. The second honeypot was focused on the actions that attackers take once they gain access to VoIP accounts. From this analysis we can conclude that phone scammers have access to more powerful technology that allows them to target a larger number of victims. They use brute force to gain access to accounts, but also obtain valid credentials through paste sites and underground markets. Different fraud campaigns have been identified through the personalized honeypot, with Toll Evasion and Revenue Share being the two most prevalent types. Fraudsters use different tricks to evade calling plan restrictions and use calls to incorrect numbers in order to check whether the compromised systems are honeypots. Directly related to this second honeypot, an architecture was deployed to deal with cybercriminals who use phone calls to carry out their scams and classify call types. A scam and spam classifier was built using different machine learning models and two chatbot models were created. The main conclusions drawn from this study are that the rules-based chatbot provides the best performance in terms of interaction time with fraudsters, and that this type of solution allows us to deplete fraudsters' resources and prevent that time from being used to attack other victims.

Regarding the third goal, which is focus on the **design, implementation and evaluation of an architecture to analyze IoT malware**, a modular framework was designed to automate the analysis of malware targeting IoT architectures. This framework supports different CPU architectures, and is capable of extracting the static and dynamic characteristics of a sample and relating it to other similar samples that have previously been analyzed. The framework was evaluated with almost 1,500 malware samples that were divided among the supported architectures. The main conclusions drawn from this goal are the following:

- The automation of malware analysis for tthe IoT is effective in performing large-scale analysis and obtaining information on which samples are similar to each other

through their clustering and visualization. This allows analysts to spend their time and effort on samples that are not similar to other known samples.

- Through static analysis it is possible to cluster malware samples, although if the characteristics are architecture dependent (e.g., ISA), it is not possible to correlate the samples between different architectures. Also, one of the main disadvantages of static features is that they can be affected by compiler options and code obfuscation.
- Through dynamic analysis it is possible to analyze samples compiled for different architectures, allowing them to be correlated with each other regardless of their architecture. However, obtaining dynamic features is more expensive than static analysis and not all architectures present in the IoT are supported for emulation in QEMU.

The fourth goal of this thesis is focus on the **development of tools and techniques to assist malware analyst and automate the process of relating malware samples**. This goal was achieved by designing a methodology to assist malware specialists in the classification, discovery and vetting of malware samples tailored to Linux-based IoT malware. Through data analysis, static and dynamic features were extracted to characterize the malware in different threats. The proposed methodology allows us to extract knowledge from large sets of related samples through the analysis of some of them and extrapolate the results to the rest. This makes it possible to investigate a large number of unlabeled samples in order to know their behavior. The main conclusions of this goal are the following:

- Malware specifically designed for the IoT is relatively simple, and techniques based on static analysis work well. This is mainly due to the fact that most samples do not use any obfuscation technique, mainly because they do not need to avoid detection by an antivirus due to the limitations of the devices in terms of memory and computing capacity.
- The level of sophistication of the malware is very varied, from families with little sophistication to some more complex ones that contain droppers or exploits.
- A large part of the analyzed clusters contain source code that is publicly available on the Internet, so code reuse leads to the creation of new malware variants, and thus very little effort is required in their creation.
- Although much of the IoT malware is aimed at creating large botnets to carry out denial of service attacks, cybercriminals are taking advantage of the resources offered by IoT systems to mine cryptocurrencies. This allows them to obtain an economic advantage without having to invest in specific hardware or pay the electricity bill.

Regarding the last proposed goal, which corresponds to **the design and implementation of a methodology for adding defensive capabilities to IoT devices**, HALE-IoT was designed. The methodology was tested on a large scale through the emulation of 395 firmware images from different vendors. Through emulation, a WAF and SSL proxy were added to support HTTPS for the devices, and the performance in terms of CPU, RAM and

## 8.2. Future Work

---

response time was measured and compared against the original firmware. The effectiveness of HALE-IoT was also evaluated with several firmware images containing some known web vulnerabilities and on four real devices. The main conclusions that can be drawn from this goal are the following:

- There are millions of IoT devices on the Internet and a large part of them are outdated or do not receive security patches to fix known issues. Of the emulated firmware images, only 21.3% have an HTTPS server by default. These HTTPS servers mostly have outdated versions of SSL/TLS and expired certificates, thus exposing this interface to Man In The Middle (MITM) attacks.
- Most devices can be non-intrusively reconfigured to isolate the original web servers to listen on localhost, replacing the original address with a WAF to inspect incoming connections and redirect them to the original web server. This allows additional security measures for web services to be incorporated into the firmware without interrupting its original operation.
- Hardening IoT devices is a promising countermeasure that allows us to reduce the attack surface of IoT devices, making it possible to detect and stop attacks against vulnerabilities that are publicly known on the Internet, with very little overhead compared with the original firmware.

## 8.2 Future Work

The work presented in this thesis is a first approach to modeling, analyzing and combating attacks in the IoT ecosystem. However, attackers are constantly evolving and looking for new ways to compromise devices. For this reason, this section presents several ways in which the work presented in this doctoral thesis could be extended and complemented.

- **Threat modeling.** In this dissertation, the Telnet, SSH and VoIP protocols have been examined, studying how these services are exploited by attackers and what actions are performed once they have control. Attackers use brute force to gain access to devices, but also exploit known vulnerabilities in other services to gain access (e.g., in HTTP, Home Network Administration Protocol (HNAP), and UPnP) [29]. However, it is not always easy to obtain exploits in the wild using traditional honeypots with a specific service as attackers can fingerprint the service. If the service matches with the model of the device to be attacked, the attacker can use the exploit to take advantage of the vulnerability. Otherwise, they take no action, making it difficult to collect new and more sophisticated attacks. This therefore, makes it difficult to obtain valuable information from the attacker since it stops the execution. Although there are some works focused on the creation of honeypots for the detection of attacks that are not easy to capture with traditional honeypots [15, 17], this line of research is still in its early days. For this reason, the field of computer security has yet to find new ways of detecting these new types of attacks, meaning that a greater research effort

is required from the scientific community in order to find more specific solutions for certain devices and versions.

- **Malware analysis.** IoT devices are a favorite target for malware creators, and Linux/IoT-based malware will continue to evolve towards similar levels of sophistication as those of other platforms such as Android or Windows. Currently, IoT malware authors do not use advanced code obfuscation techniques to complicate static analysis, and mainly rely on packaging samples with Ultimate Packer for eXecutables (UPX) [73, 74] or some modification of it (for example, inserting garbage bytes, modifying strings or magic headers) to prevent it from being unpacked directly with the official tool. Although the proportion of packed IoT malware in the wild is still small, compared with the total number of samples, our automatic analysis framework does not unpack them, and is therefore less robust against code obfuscation. In addition, IoT malware can use sandbox detection or anti-debugging techniques to detect whether the sample is running in a controlled environment [18], allowing it to detect whether it is being monitored by tools that use technologies similar to a code debugger or if the sample is running in an environment that uses virtualization or emulation technologies. Therefore, our malware analysis framework can be extended to use custom kernel environments that allow instrumentation and monitoring, thus making detection by malware more difficult. Furthermore, the work can be complemented via new code similarity metrics that use higher level features such as decompiled code, and thus allow searching for code similarities between samples compiled for different architectures.
- **Retrofit security measures.** In this doctoral thesis, we have designed a methodology for implementing defensive measures in the firmware of legacy devices that are no longer maintained by the manufacturer and therefore lack software updates. There are different areas of interest for future work in this direction. First of all, our methodology is focused on Type-I: general purpose OS-based devices according to the taxonomy defined in [75]. The application of defensive measures to other types of targets that are more limited in terms of storage and memory needs further investigation. During the evaluation of our methodology, we did not focus on performing extensive regression testing on the web interfaces of the device or other services. Therefore, it would be useful to carry out a large-scale functional analysis of the impact that the implementation of defensive systems in devices can have. In addition, in the Hale-IoT evaluation we used the Firmadyne dataset [26], which is considered the state of the art. However, being proprietary firmware, the Firmadyne dataset only contains the download links and not the firmware images. At the time of the experiments, only 4,809 out of 23,035 images were available. Therefore, future efforts by the community are needed to create datasets that are representative (i.e., with different CPU architectures, OS's, and functionalities), do not have licensing issues, and are always available.



# Bibliography

- [1] Knud Lasse Lueth. IoT Analytics, “State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time,” 2020, Last accessed on July 2022. [Online]. Available: <https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/>
- [2] Mohammad Hasan. Iot Analytics, “State of iot 2022: Number of connected IoT devices growing 18% to 14.4 billion globally,” 2022, Last accessed on July 2022. [Online]. Available: <https://iot-analytics.com/number-connected-iot-devices/>
- [3] The OWASP IoT Security Team, “OWASP Internet of Things Project - OWASP,” 2019, Last accessed on July 2022. [Online]. Available: [https://wiki.owasp.org/index.php/OWASP\\_Internet\\_of\\_Things\\_Project](https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project)
- [4] M. Janiszewski, M. Rytel, P. Lewandowski, and H. Romanowski, “Variot - vulnerability and attack repository for the internet of things,” in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 752–755.
- [5] E. Kovacs, “Volume of Attacks on IoT/OT Devices Increasing: Microsoft Study,” 2021, Last accessed on July 2022. [Online]. Available: <https://www.securityweek.com/volume-attacks-iotot-devices-increasing-microsoft-study>
- [6] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, “Understanding the mirai botnet,” in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>
- [7] AV-TEST - The Independent IT-Security Institute, “AV-ATLAS,” 2022, Last accessed on Jul. 2022. [Online]. Available: <https://portal.av-atlas.org/>
- [8] A. Costin, A. Zarras, and A. Francillon, “Automated dynamic firmware analysis at scale: a case study on embedded web interfaces,” in *11th ACM on Asia Conference on Computer and Communications Security*, 2016.

## Bibliography

---

- [9] R. P. GmbH, “Inside a PBX - Discovering a Firmware Backdoor,” Last accessed on Aug. 2022. [Online]. Available: <https://blog.redteam-pentesting.de/2021/inside-a-pbx/>
- [10] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, “A survey on honeypot software and data analysis,” *arXiv preprint arXiv:1608.06249*, 2016.
- [11] W. Fan, Z. Du, D. Fernández, and V. A. Villagrà, “Enabling an anatomic view to investigate honeypot systems: A survey,” *IEEE Systems Journal*, vol. 12, no. 4, pp. 3906–3919, 2018.
- [12] H. Šemić and S. Mrdovic, “Iot honeypot: A multi-component solution for handling manual and mirai-based attacks,” in *2017 25th Telecommunication Forum (TELFOR)*, 2017, pp. 1–4.
- [13] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, “Iotpot: Analysing the rise of iot compromises,” in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.
- [14] M. Wang, J. Santillan, and F. Kuipers, “Thingpot: an interactive internet-of-things honeypot,” *arXiv preprint arXiv:1807.04114*, 2018.
- [15] T. Luo, Z. Xu, X. Jin, Y. Jia, and X. Ouyang, “Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices,” *Black Hat*, vol. 2017, 2017.
- [16] A. Tambe, Y. L. Aung, R. Sridharan, M. Ochoa, N. O. Tippenhauer, A. Shabtai, and Y. Elovici, “Detection of threats to iot devices using scalable vpn-forwarded honeypots,” in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 85–96. [Online]. Available: <https://doi.org/10.1145/3292006.3300024>
- [17] A. Vetterl and R. Clayton, “Honware: A virtual honeypot framework for capturing cpe and iot zero days,” in *2019 APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 2019, pp. 1–13.
- [18] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, “Understanding linux malware,” in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 161–175.
- [19] A. Costin and J. Zaddach, “IoT malware: Comprehensive survey, analysis framework and case studies,” 2018.
- [20] K. Monnappa, “Automating linux malware analysis using limon sandbox,” *Black Hat Europe*, vol. 2015, pp. IV–A, 2015.
- [21] A. Darki, C.-Y. Chuang, M. Faloutsos, Z. Qian, and H. Yin, “Rare: A systematic augmented router emulation for malware analysis,” in *International Conference on Passive and Active Network Measurement*. Springer, 2018, pp. 60–72.



- 
- [22] A. Darki and M. Faloutsos, "Riotman: A systematic analysis of iot malware behavior," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 169–182. [Online]. Available: <https://doi.org/10.1145/3386367.3431317>
- [23] V. Iyengar, M. Koser, R. Binjve, and A. Gat, "Detux: The Multiplatform Linux Sandbox," 2016. [Online]. Available: <https://github.com/detuxsandbox/detux>
- [24] The Cuckoo Team, "Cuckoo Sandbox - Automated Malware Analysis," 2018. [Online]. Available: <https://cuckoosandbox.org/>
- [25] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares," in *23rd USENIX Security Symposium*, 2014.
- [26] D. Chen, M. Egele, M. Woo, and D. Brumley, "Towards Automated Dynamic Analysis for Linux-based Embedded Firmware," in *NDSS*, vol. 25, 2016.
- [27] M. Kim, D. Kim, E. Kim, S. Kim, Y. Jang, and Y. Kim, "Firmae: Towards large-scale emulation of iot firmware for dynamic analysis," in *Annual Computer Security Applications Conference*, ser. ACSAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 733–745. [Online]. Available: <https://doi.org/10.1145/3427228.3427294>
- [28] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019.
- [29] B. Zhao, S. Ji, W.-H. Lee, C. Lin, H. Weng, J. Wu, P. Zhou, L. Fang, and R. Beyah, "A large-scale empirical study on the vulnerability of deployed iot devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 3, pp. 1826–1840, 2022.
- [30] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: An analysis of IoT devices on home networks," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1169–1185. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity19/presentation/kumar-deepak>
- [31] A. Cui and S. 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*, ser. ACSAC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 97–106. [Online]. Available: <https://doi.org/10.1145/1920261.1920276>
- [32] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 636–654.

## Bibliography

---

- [33] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Security implications of permission models in smart-home application frameworks," *IEEE Security & Privacy*, vol. 15, no. 2, pp. 24–30, 2017.
- [34] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "FlowFence: Practical data protection for emerging IoT application frameworks," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 531–548. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/fernandes>
- [35] X. Feng, R. Sun, X. Zhu, M. Xue, S. Wen, D. Liu, S. Nepal, and Y. Xiang, "Snipuzz: Black-box fuzzing of iot firmware via message snippet inference," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 337–350. [Online]. Available: <https://doi.org/10.1145/3460120.3484543>
- [36] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "Ddos-capable iot malwares: Comparative analysis and mirai investigation," *Security and Communication Networks*, vol. 2018, 2018.
- [37] P. Celeda, R. Krejci, J. Vykopal, and M. Drasar, "Embedded malware-an analysis of the chuck norris botnet," in *2010 European Conference on Computer Network Defense*. IEEE, 2010, pp. 3–10.
- [38] A. Wang, R. Liang, X. Liu, Y. Zhang, K. Chen, and J. Li, "An inside look at iot malware," in *International Conference on Industrial IoT Technologies and Applications*. Springer, 2017, pp. 176–186.
- [39] M. Bohio, "Analyzing a backdoor/bot for the mips platform," *SANS Institute, Tech. Rep.*, 2015.
- [40] S. Edwards and ioannis Profetis, "Hajime: Analysis of a decentralized internet worm fot IoT devices," Rapidity Networks, Tech. Rep., 2016. [Online]. Available: <https://security.rapiditynetworks.com/publications/2016-10-16/hajime.pdf>
- [41] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, "Measurement and analysis of hajime, a peer-to-peer iot botnet," *Network and Distributed System Security (NDSS) Symposium*, 2019. [Online]. Available: [https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019\\_02B-3\\_Herwig\\_paper.pdf](https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_02B-3_Herwig_paper.pdf)
- [42] M. Alhanahnah, Q. Lin, Q. Yan, N. Zhang, and Z. Chen, "Efficient signature generation for classifying cross-architecture iot malware," in *2018 IEEE Conference on Communications and Network Security (CNS)*, May 2018, pp. 1–9.
- [43] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, "Lightweight classification of iot malware based on image recognition," in *2018 IEEE 42nd Annual*

- Computer Software and Applications Conference (COMPSAC)*, vol. 02, July 2018, pp. 664–669.
- [44] K. D. T. Nguyen, T. M. Tuan, S. H. Le, A. P. Viet, M. Ogawa, and N. L. Minh, “Comparison of three deep learning-based approaches for iot malware detection,” in *2018 10th International Conference on Knowledge and Systems Engineering (KSE)*, 2018, pp. 382–388.
  - [45] A. Anwar, H. Alasmarty, J. Park, A. Wang, S. Chen, and D. Mohaisen, “Statically dissecting internet of things malware: Analysis, characterization, and detection,” in *Information and Communications Security*, W. Meng, D. Gollmann, C. D. Jensen, and J. Zhou, Eds. Cham: Springer International Publishing, 2020, pp. 443–461.
  - [46] H. Alasmarty, A. Khormali, A. Anwar, J. Park, J. Choi, A. Abusnaina, A. Awad, D. Nyang, and A. Mohaisen, “Analyzing and detecting emerging internet of things malware: A graph-based approach,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8977–8988, 2019.
  - [47] H. Haddadjouh, A. Mohtadi, A. Dehghantanaha, H. Karimipour, X. Lin, and K.-K. R. Choo, “A multikernel and metaheuristic feature selection approach for iot malware threat hunting in the edge layer,” *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4540–4547, 2021.
  - [48] H. Darabian, A. Dehghantanha, S. Hashemi, S. Homaoun, and K.-K. R. Choo, “An opcode-based technique for polymorphic internet of things malware detection,” *Concurrency and Computation: Practice and Experience*, vol. 32, no. 6, p. e5173, 2020.
  - [49] T. N. Phu, K. H. Dang, D. N. Quoc, N. T. Dai, and N. N. Binh, “A novel framework to classify malware in mips architecture-based iot devices,” *Security and Communication Networks*, vol. 2019, 2019.
  - [50] J. Jeon, J. H. Park, and Y.-S. Jeong, “Dynamic analysis for iot malware detection with convolution neural network model,” *IEEE Access*, vol. 8, pp. 96 899–96 911, 2020.
  - [51] F. Shahzad, M. Shahzad, and M. Farooq, “In-execution dynamic malware analysis and detection by mining information in process control blocks of linux os,” *Inf. Sci.*, vol. 231, p. 45–63, may 2013. [Online]. Available: <https://doi.org/10.1016/j.ins.2011.09.016>
  - [52] R. Isawa, T. Ban, Y. Tie, K. Yoshioka, and D. Inoue, “Evaluating disassembly-code based similarity between iot malware samples,” in *2018 13th Asia Joint Conference on Information Security (AsiaJCS)*, 2018, pp. 89–94.
  - [53] M. Oosterhof, “Cowrie SSH and Telnet Honeypot,” 2015, Last accessed on July 2022. [Online]. Available: <https://www.cowrie.org/>
  - [54] F. Bellard, “Qemu, a fast and portable dynamic translator,” in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '05. USA: USENIX Association, 2005, p. 41.

## Bibliography

---

- [55] A. L. Johnson, "Endpoint Protection - Symantec Enterprise," 2016, Last accessed on July 2022. [Online]. Available: <https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=ed1463a7-41ed-4b2f-adfd-97ccacb9d593&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68>
- [56] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive datasets*. Cambridge university press, 2014.
- [57] M. Sahin, A. Francillon, P. Gupta, and M. Ahamad, "Sok: Fraud in telephony networks," in *2017 IEEE European Symposium on Security and Privacy (EuroSP)*, 2017, pp. 235–250.
- [58] T. Bocklisch, J. Faulkner, N. Pawlowski, and A. Nichol, "Rasa: Open source language understanding and dialogue management," 2017.
- [59] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.
- [60] R. Team, *Radare2 Book*. GitHub, 2017. [Online]. Available: <https://radare.gitbooks.io/radare2book/content/>
- [61] J. Carrillo-Mondéjar, J. M. Castelo Gómez, J. Roldán-Gómez, Sergio Ruiz Villafranca and G. Suárez-Tangil, "Hajime's Return: Stories from a Customized Honeypot for IoT," *Journal of Internet Technology*, 2022, [Under review].
- [62] J. Carrillo-Mondéjar, J. Martinez, and G. Suarez-Tangil, "On how voip attacks foster the malicious call ecosystem," *Computers & Security*, vol. 119, p. 102758, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167404822001535>
- [63] J. Carrillo-Mondéjar, J. Martinez, and G. Suarez-Tangil, "A chatbot for selectively engaging with online fraudsters," *IEEE Access*, 2022, [Under review].
- [64] J. Carrillo-Mondejar, J. M. Castelo Gomez, C. Núñez-Gómez, J. Roldán Gómez, and J. L. Martínez, "Automatic analysis architecture of iot malware samples," *Security and Communication Networks*, vol. 2020, p. 8810708, Oct 2020. [Online]. Available: <https://doi.org/10.1155/2020/8810708>
- [65] J. Carrillo-Mondéjar, J. Martínez, and G. Suarez-Tangil, "Characterizing linux-based malware: Findings and recent trends," *Future Generation Computer Systems*, vol. 110, pp. 267–281, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X19325002>

- [66] J. Carrillo-Mondéjar, Hannu Turtiainen, Andrei Costin, J.L. Martínez, and G. Suarez-Tangil, "Hale-iot: Hardening legacy internet-of-things devices via retrofitting defensive firmware modifications and implants," *Internet of Things Journal*, 2022, [Minor revision].
- [67] J. M. Castelo Gómez, J. Roldán Gómez, J. Carrillo Mondéjar, and J. L. Martínez Martínez, "Non-volatile memory forensic analysis in windows 10 iot core," *Entropy*, vol. 21, no. 12, 2019. [Online]. Available: <https://www.mdpi.com/1099-4300/21/12/1141>
- [68] J. M. Castelo Gómez, J. Carrillo Mondéjar, J. Roldán Gómez, and J. L. Martínez Martínez, "A context-centered methodology for IoT forensic investigations," *International Journal of Information Security*, Nov. 2020. [Online]. Available: <https://doi.org/10.1007/s10207-020-00523-6>
- [69] J. M. Castelo Gómez, J. Carrillo Mondéjar, J. Roldán Gómez, and J. Martínez Martínez, "Developing an iot forensic methodology. a concept proposal," *Forensic Science International: Digital Investigation*, vol. 36, p. 301114, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2666281721000081>
- [70] J. M. Castelo Gómez, J. Carrillo-Mondéjar, J. L. Martínez Martínez, and J. Navarro García, "Forensic analysis of the xiaomi mi smart sensor set," *Forensic Science International: Digital Investigation*, vol. 42, p. 301451, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2666281722001329>
- [71] J. Carrillo-Mondéjar, J. M. Castelo-Gómez, J. Roldán-Gómez, and J. L. Martínez, "An instrumentation based algorithm for stack overflow detection," *Journal of Computer Virology and Hacking Techniques*, vol. 16, no. 3, pp. 245–256, Sep 2020. [Online]. Available: <https://doi.org/10.1007/s11416-020-00359-7>
- [72] J. Roldán-Gómez, J. Boubeta-Puig, J. M. Castelo Gómez, J. Carrillo-Mondéjar and J. L. Martínez Martínez, "Attack pattern recognition in the Internet of Things using complex event processing and machine learning," in *2021 IEEE International Conference on Systems, Man, and Cybernetics*, Oct. 2021.
- [73] A. A. Al Alsadi, K. Sameshima, J. Bleier, K. Yoshioka, M. Lindorfer, M. van Eeten, and C. H. Gañán, "No spring chicken: Quantifying the lifespan of exploits in iot malware using static and dynamic analysis," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 309–321. [Online]. Available: <https://doi.org/10.1145/3488932.3517408>
- [74] O. Alrawi, C. Lever, K. Valakuzhy, R. Court, K. Snow, F. Monroe, and M. Antonakakis, "The circle of life: A Large-Scale study of the IoT malware lifecycle," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association,

## Bibliography

---

Aug. 2021, pp. 3505–3522. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity21/presentation/alrawi-circle>

- [75] M. Muench, J. Stijohann, F. Kargl, A. Francillon, and D. Balzarotti, “What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices,” in *NDSS*, 2018. [Online]. Available: [http://s3.eurecom.fr/docs/ndss18\\_muench.pdf](http://s3.eurecom.fr/docs/ndss18_muench.pdf)