On Leveraging Next-Generation Deep Learning Techniques for IoT Malware Classification, Family Attribution and Lineage Analysis

Mirabelle Dib

A Thesis

in

The Department of Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Computer Science

Concordia University

Montréal, Québec, Canada

December 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mirabelle Dib**

Entitled: **On Leveraging Next-Generation Deep Learning Techniques for IoT Malware Classification, Family Attribution and Lineage Analysis**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Tristan Glatard

_____ Examiner
Dr. Amr Youssef

_____ Examiner
Dr. Tristan Glatard

_____ Thesis Co-Supervisor
Dr. Chadi Assi

_____ Thesis Co-Supervisor
Dr. Elias Bou-Harb

Approved by _____
Leila Kosseim, Director
The Department of Computer Science and Software Engineering

December 17, 2021

_____
Mourad Debbabi, Dean,
Gina Cody School of Engineering and Computer Science

Abstract

On Leveraging Next-Generation Deep Learning Techniques for IoT Malware Classification, Family Attribution and Lineage Analysis

Mirabelle Dib

Recent years have witnessed the emergence of new and more sophisticated malware targeting insecure Internet of Things (IoT) devices, as part of orchestrated large-scale botnets. Moreover, the public release of the source code of popular malware families such as `Mirai` [1] has spawned diverse variants, making it harder to disambiguate their ownership, lineage, and correct label. Such a rapidly evolving landscape makes it also harder to deploy and generalize effective learning models against retired, updated, and/or new threat campaigns. To mitigate such threat, there is an utmost need for effective IoT malware detection, classification and family attribution, which provide essential steps towards initiating attack mitigation/prevention countermeasures, as well as understanding the evolutionary trajectories and tangled relationships of IoT malware. This is particularly challenging due to the lack of fine-grained empirical data about IoT malware, the diverse architectures of IoT-targeted devices, and the massive code reuse between IoT malware families.

To address these challenges, in this thesis, we leverage the general lack of obfuscation in IoT malware to extract and combine static features from multi-modal views of the executable binaries (e.g., images, strings, assembly instructions), along with Deep Learning (DL) architectures for effective IoT malware classification and family attribution. Additionally, we aim to address concept drift and the limitations of inter-family classification due to the evolutionary nature of IoT malware, by detecting in-class evolving IoT malware variants and interpreting the meaning behind their mutations. To this end, we perform the following to achieve our objectives:

First, we analyze 70,000 IoT malware samples collected by a specialized IoT honeypot and popular malware repositories in the past 3 years. Consequently, we utilize features extracted from

strings- and image-based representations of IoT malware to implement a multi-level DL architecture that fuses the learned features from each sub-component (i.e, images, strings) through a neural network classifier. Our in-depth experiments with four prominent IoT malware families highlight the significant accuracy of the proposed approach (99.78%), which outperforms conventional single-level classifiers, by relying on different representations of the target IoT malware binaries that do not require expensive feature extraction. Additionally, we utilize our IoT-tailored approach for labeling unknown malware samples, while identifying new malware strains.

Second, we seek to identify when the classifier shows signs of aging, by which it fails to effectively recognize new variants and adapt to potential changes in the data. Thus, we introduce a robust and effective method that uses contrastive learning and attentive Transformer models to learn and compare semantically meaningful representations of IoT malware binaries and codes without the need for expensive target labels. We find that the evolution of IoT binaries can be used as an augmentation strategy to learn effective representations to contrast (dis)similar variant pairs. We discuss the impact and findings of our analysis and present several evaluation studies to highlight the tangled relationships of IoT malware, as well as the efficiency of our contrastively learned fine-grained feature vectors in preserving semantics and reducing out-of-vocabulary size in cross-architecture IoT malware binaries.

We conclude this thesis by summarizing our findings and discussing research gaps that lay the way for future work.

Dedication

This thesis is dedicated to my parents, Eliane and Toufic.

For their endless love, support and encouragement

Acknowledgments

I wish to thank my supervisor Prof. Chadi Assi for his dedicated support, guidance and patience throughout my Masters degree. I would also like to thank my co-supervisor Dr. Elias Bou-Harb, for his invaluable feedback, perspective and positive criticism of my research experiments.

My gratitude extends to Prof. Chadi Assi and the Faculty of Engineering and Computer Science for the funding opportunity to complete my studies at the Department of Computer Science and Software Engineering.

Additionally, I would like to thank the committee members who were more than generous with their expertise and precious time, and offered me valuable comments towards improving my work.

My thanks extends to Prof. Nizar Bouguila for sharing his expertise and providing generous feedback. I would also like to thank Dr. Aiman Hanna for constantly granting me teaching opportunities that helped me financially and ignited my passion for teaching.

Finally, I would like to thank my fellow colleague Dr. Sadegh Torabi for his insightful comments and collaboration at every stage of my research projects.

"If we knew what it was we were doing, it would not be called research, would it?"

— Albert Einstein

Contribution of Authors

First Contribution (Accepted in TNSM'21): Mirabelle Dib and Sadegh Torabi equally conceived the presented idea. Mirabelle developed the theory, carried out the experiments, and took the lead in writing the manuscript. Sadegh Torabi, Elias Bou-Harb and Chadi Assi provided critical feedback and helped shape the research, analysis and manuscript.

Citation: Dib, Mirabelle, Sadegh Torabi, Elias Bou-Harb, and Chadi Assi. "A Multi-Dimensional Deep Learning Framework for IoT Malware Classification and Family Attribution." IEEE Transactions on Network and Service Management (2021).

Second Contribution (Submitted at ACM ASIACCS'22): Mirabelle Dib solely conceived the presented idea. Mirabelle designed, planned and executed the experiments. Mirabelle took the lead in writing the paper. Sadegh Torabi, Elias Bou-Harb and Chadi Assi discussed the results and provided comments on the final version of the manuscript. Nizar Bouguila shared his expertise and opinion on the presented theory.

Citation: Dib, Mirabelle, Sadegh Torabi, Elias Bou-Harb, Nizar Bouguila and Chadi Assi. "EVOLIoT: A Self-Supervised Contrastive Learning Framework for Detecting and Characterizing Evolving IoT Malware Variants". In Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security. (2022)

Contents

List of Figures

List of Tables

List of Abbreviations

AV	Anti Virus
C&C	Command & Control
CFG	Control Flow Graph
CSSL	Contrastive Self-Supervised Learning
DAG	Directed Acyclic Graph
DDoS	Distributed Denial of Service
DL	Deep Reinforcement Learning
ELF	Executable and Linkable Format
GOT	Global Offset Table
IoT	Internet of Things
LSTM	Long Short-Term Memory
ML	Machine Learning
MST	Minimum Spanning Tree
NLP	Natural Language Processing
NN	Neural Network
OS	Operating System
PE	Portable Executable
UPX	Ultimate Packer for eXecutables

VT VirusTotal

Chapter 1

Introduction

1.1 Problem Scope and Motivation

The emergence of Internet of Things malware, which leverages exploited devices to perform large-scale attacks (e.g., `Mirai` botnet [1]), presents a major threat to the Internet ecosystem [2, 7]. To mitigate such threat, there is an utmost need to evaluate the security of IoT paradigm by developing effective learning-based tools and techniques for the prompt detection, classification and characterization of evolving IoT threats. This is challenging due to: (i) the world-wide spread of a large number of interconnected insecure IoT devices, (ii) the lack of fine-grained IoT malware labels and well-designed ground truth datasets for evaluation, (iii) the complex relationship and similarities in terms of code reuse among emerging malware variants and (iv) the lack of scalable family attribution, binary similarity and lineage inference tools that support multiple CPU architectures.

We present in the following sub-sections our objectives and how we aim at addressing these challenges by leveraging deep learning techniques along with static malware analysis to classify, detect and investigate IoT malware variants and their changing malicious nature.

1.2 Objectives and Research Questions

The main objective of this thesis is to leverage next-generation techniques to implement effective, scalable and robust IoT-tailored approaches for correctly classifying IoT malware binaries,

detecting evolving variants over time, and interpreting the meaning behind their drift. Given the threats associated with the rise of IoT malware strains and the challenges associated with the lack of empirical data and representative labels, the capability to combat concept drift and the limitations of intra-family classification by inferring with certainty to which family a malware belongs and highlighting the dynamics behind the emergence of new malware strains over time, is essential to build a better understanding about the changing threat landscape while supporting the development of effective detection and mitigation measures.

In particular, we want to answer the following research questions (RQs):

(1) *How can we develop and evaluate a classification approach that automatically learns static features from multiple raw modalities of IoT malware binaries, while improving the overall classification accuracy and reducing the cost of artificial feature engineering? Given the benefits of the proposed multi-level deep learning approach, how can we leverage it to detect new or unknown malware samples given the information about existing malware families?*

(2) *Is the performance of classifiers likely to degrade with time, as the nature of the IoT malware landscape is changing? How can we leverage contrastive learning and attentive Transformer models to detect in-class evolving IoT malware binaries and understand the meaning behind the changing relationship between them by using an interpretable strings-based similarity analysis?*

1.3 Contributions

We present a summary of the main contributions made throughout this thesis to answer the aforementioned objectives:

1.3.1 A Multi-Dimensional Deep Learning Framework for IoT Malware Classification and Family Attribution

We propose a multi-level approach that leverages a combination of static features along with DL techniques for effective IoT malware classification and family attribution. Our aim is to (i)

leverage the lack of widely deployed sophisticated malware obfuscation for extracting static features from different modalities of the IoT malware binaries, (ii) overcome the limitations of artificial feature engineering by leveraging deep learning methods that automatically and efficiently extract strings features, and (iii) improve the overall classification accuracy by building a multi-dimensional architecture that combines different representations of the target IoT malware binaries.

To achieve our objectives and answer our first research question, we leverage 70,000 IoT malware executables detected by a specialized IoT honeypot (IoTPOT [8]) and publicly available threat repositories (e.g., VirusTotal [9]) during the past three years. Additionally, we devise an image-based analysis technique to visualize malware binaries files as images and we utilize reverse-engineering and static malware analysis techniques to extract useful strings from the binaries.

More importantly, we implement our multi-level DL architecture by fusing the learned features from each image-based and strings-based component through a neural network classifier. Consequently, we perform a series of experiments and evaluate the effectiveness of our approach in comparison to state-of-the-art approaches. In addition to classifying known IoT malware, we leverage the proposed approach for classifying unknown samples, which are attributed to a few predominant IoT malware families.

The outcome of this contribution is published/presented at the IEEE TNSM (2021) [10].

1.3.2 EVOLIoT: A Self-Supervised Contrastive Learning Framework for Detecting and Characterizing Evolving IoT Malware Variants

The increasing number of detected IoT malware and the threat associated with the IoT-driven cyber attacks has pushed the security community to develop rigorous mitigation approaches against the spread of IoT malware strains by building effective learning-based malware detectors and classifiers. Yet, it is still unclear what makes each group of malware distinct or how the same family evolves over time. Our aim is to understand the complex relationship and (dis)similarities among malware variants by (i) understanding how the performance of classifiers degrade with time as new malware campaigns are introduced, (ii) detecting drifting IoT malware variants within the same malware family and consequently (iii) interpreting the meaning behind their drift.

To reach our objectives and thus answer our second research question, we adopt a cross-architecture

code-based analysis that can capture a binary's malicious intent and evolutionary essence regardless of its instruction set architecture (ISA). We propose a self-supervised contrastive learning approach based on pre-trained attentive language models such as BERT [11], which effectively learns and compares semantically meaningful representations of binary code, without the need for expensive target labels. We evaluate our approach on the same large corpus of IoT malware binaries used in the previous work, and highlight the constant evolution of variants among each family from different perspectives. We also extensively evaluate our proposed method on different applications and test our well-balanced instruction normalization strategy to demonstrate its effectiveness in conserving as much contextual/semantic information as possible for cross-architecture syntactic variations.

The outcome of this contribution is submitted at the ACM ASIACCS Conference (2022).

1.4 Thesis Organization

The remainder of this thesis is structured as follows: In Chapter 2, we present background information along with a review of related work. In Chapter 3, we present our multi-dimensional approach for classifying and attributing IoT malware families. Chapter 4 presents our self-supervised contrastive learning approach towards detecting in-class evolving IoT malware variants and characterizing their mutations.

Chapter 2

Background and Related Work

In what follows, we provide background information about the IoT threat landscape, the ELF binary structure and common malware analysis techniques, followed by a review of literature with respect to various concerned topics as presented in recent published work, such as IoT malware detection and classification, malware evolution and concept drift. We also provide an overview of the dataset that we leverage throughout the presented thesis contributions.

2.1 IoT Threat Landscape

2.1.1 Demystifying IoT Cyber Attacks

The insecurities of Internet-connected IoT devices have made them vulnerable to hijacking and weaponization for use en masse to carry out cyber attacks. Under the control of a botnet called `Mirai` [1] which took the Internet by storm in late 2016, 600,000 vulnerable IoT devices overwhelmed several high-profile targets, such as Dyn (major US domain service company), OVH (cloud computing company), Airbnb, Github, and Twitter among others [2]. Such attack served as an indication of the potential devastating impact that these exploited devices represent. Since then, the volume and sophistication of attacks targeting IoT devices have grown abruptly, and new IoT-tailored malware/botnets are frequently emerging [2, 7]. The release of the `Mirai` botnet source code has fueled the rapid evolution of more advanced and sophisticated `Mirai`-like variants such

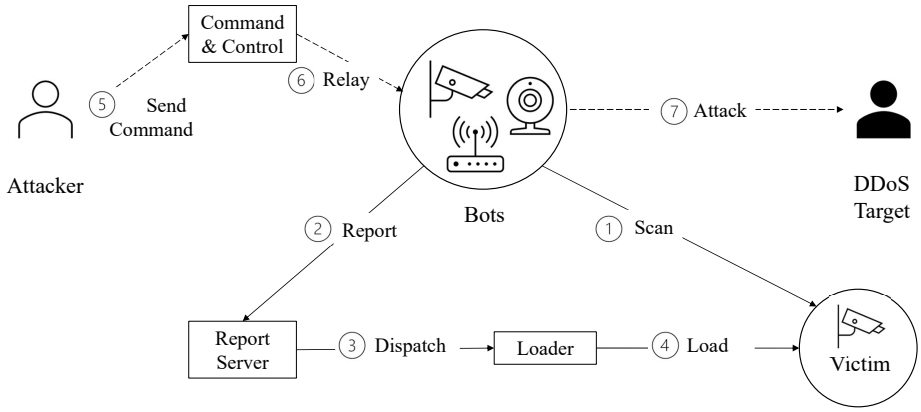


Figure 2.1: The overall `Mirai` botnet operation [2].

as `Hajime` [12], `Satori` [13], and `BrickerBot` [14] to name a few, which follow the same overall operation as presented in Figure 2.1 to find and exploit vulnerable devices. In general, `Mirai` bots start by scanning the IPV4 address space for other devices that run telnet or SSH, attempting to brute-force log in using 10 usernames and passwords pairs randomly selected from a hardcoded dictionary of 62 credentials (①). After a successful login, the victim IP and valid credentials are reported back to a fixed server (②), which in turn asynchronously triggers a loader program (③) to download and execute an architecture-specific malicious payload (④). After infection, `Mirai` fortifies itself by killing other processes associated with competing variants like `Qbot` [15] and concealing its presence by deleting the downloaded binary. Meanwhile, the bot listens for attack commands from its command and control (C2) server while concurrently scanning for new victims (⑤, ⑥). Now in contrast to `Mirai` which scans services such as HTTP and Telnet on TCP ports 80/23/2323, the analysis of recent `Mirai`-like variants such as `Satori` highlighted scanning activities towards TCP port 37215 to execute remote code into the device upgrade process of specific Huawei routers. Furthermore, a spike in exploits against TCP port 52869 indicates a clear targeted attack towards Huawei routers. Moreover, other `Mirai`-related botnets have started leveraging not only telnet credentials brute-forcing but also exploiting very specific software vulnerabilites in IoT device firmware [16,17]. This shows the dedication of cybercriminals to building more sophisticated botnets, and hence is a revelation of a real evolution from the first `Mirai` botnet.

2.1.2 IoT Security and Vulnerability Assessment

The "s" in IoT stands for security. In other words, a competitive landscape (i.e., profiting businesses, short time-to-market) and technical constraints on IoT devices (i.e. low-cost, limited computational powers) have made it challenging for IoT devices' manufacturers to design and implement complex security features in these devices. Consequently, this negligence of several security considerations have enabled the rise of IoT-driven cyber attacks. An extensive survey by Neshenko et al. [18] on several studies dedicated to IoT (in)security have revealed a total of nine (9) degrees of IoT vulnerabilities. Of which:

Weak Authentication and Insufficient Access Control. The main goal of attacks against authentication and access control is to gain unauthorized access to IoT resources and data to further perform malicious actions. The most common type of attack is a *dictionary attack*, in which a botnet attempts to brute force the victim credentials. On this subject, Kolias et al. [19] has explained how a dictionary attack can jeopardize and manipulate millions of Internet-facing IoT devices into launching coordinated attacks against important services. The `Mirai` botnet has introduced a balanced dictionary attack in which a random subset of the most frequently used credentials is picked and attempted [2]. Such attacks are enabled by a majority of devices allowing passwords of insufficient complexity and length, such as "admin" or "1234" [20]. Moreover, after installation, many devices do not request its consumers to change the default user credentials. Markowsky et al. [21] have concluded that the Carna botnet has surveyed the entire IPv4 address space in 2012 and unveiled more than 1.6 million devices around the world using user default credentials. Cui et al. [22] have performed large-scale Internet scans of IoT devices and found over half a million publicly accessible IoT devices configured with factory default root password. A study by Alrawi et al. [23] has identified over 39 types of IoT devices with elevated permissions which could allow unauthorized users to gain privilege access and spy on the end-users.

Networks-based Vulnerabilities. Various studies have focused on IoT-specific vulnerabilities created by network or protocol weaknesses. For example, the security of the ZigBee protocol built to address the unique needs of low-cost, low-power wireless IoT networks has been studied by several works [24]. The authors have demonstrated how ZigBee-enabled IoT devices can be compromised,

controlled and manipulated to conduct denial of service on IoT. They pinpointed an insufficient property of key management where the keys are transmitted unencrypted hence enabling the leak of sensitive information and allowing control over the devices.

Unnecessary Open Ports. Another network vulnerability to be considered is related to port blocking policies. By conducting port scanning, penetration testing and fingerprinting examinations of numerous consumer IoT devices, Sachidananda et al. [25] have discovered that a large number of devices have unnecessary open ports/services (e.g., SSH(22), Telnet(23), HTTP(80)), which could be easily leveraged to leak confidential information about operating systems, device types, IP addresses, and transferred data. For instance, the Belkin smart camera was found to expose a large number of ports, 5 TCP and 31 UDP. Another example is the HP printer which responds to a special port 9100 which was used for printing with no authorization. Such vulnerability was exploited to attack more than 150,000 printers [26, 27].

Firmware-based Vulnerabilities. Studies [16, 17] have shown that IoT botnets started leveraging not only telnet credentials brute-forcing but also exploiting a myriad of software vulnerabilities (e.g., backdoors, primary root access points, and a lack of Secure Socket Layer (SSL) usage [18,28]) in IoT device firmware. Such flaws have allowed attackers to steal WiFi credentials [29], control smart TVs and home assist devices [30], and turn smart thermostats into spy gadgets [31]. Costin et al. [32] conducted a large-scale assessment of IoT device firmware and discovered a total of 38 previously unknown vulnerabilities in over 693 firmware images, altogether affecting 140K accessible devices over the Internet. In addition, Konstantinou et al. [33] revealed the drastic potential of vulnerable power grid firmware to corrupt traffic signals and cause uncontrolled power outages.

Improper Patch Management Capabilities. To minimize such attack vectors, IoT operating systems and firmware should be regularly updated/patched. Nevertheless, these mechanisms are not widely adopted in IoT devices as many manufacturers do not have set up automated patch-update mechanisms. Tekeoglu et al. [28] identified a vulnerability in available update/patch mechanisms that lack integrity guarantees, rendering them susceptible to being maliciously modified and applied at large.

2.1.3 IoT Malware Data Collection

Studying the threat landscape of smart IoT devices is challenging given the rareness of IoT-related empirical data and the fact that IoT encompasses an array of different types of devices, that can be deployed in a large variety of environments. Motivated by this, efforts have been put to collect IoT-tailored data, particularly malware binaries. For instance, IoTPOT offered by Pa et al. [34] was the first of its kind honeypot emulating Telnet services of various IoT devices running on different CPU architectures. They observed multiple attempts to download external malware binary files, as well as three phases of Telnet-based attacks, namely, intrusion, infection and monetization. Similarly, Guarnizo et al. [35] presented SIPHON, the Scalable high-Interaction Honeypt platform for IoT devices. The authors demonstrated how worldwide wormholes and a few physical devices can mimic various IoT devices and attract malicious traffic, such as popular target locations, scanned ports, and user agents. Gandhi et al. [36] proposed another IoT-honeypot called HIoTPOT and observed that 67% of daily connections were unauthorized, which confirms the increasing interest of attackers in finding vulnerable IoT devices.

In an alternative work, Dowling et al. [37] designed a honeypot which explores attacks against ZigBee-based IoT devices by emulating a ZigBee gateway. The authors reported that 94% of all attacks were dictionary attacks. Luo et al. [38] leveraged machine learning techniques to automatically learn the behaviors of different types of IoT devices and create an intelligent and interactive honeypot.

By simulating a variety of IoT devices, the proposed honeypots have addressed the lack of empirical knowledge/data about IoT by facilitating the collection of IoT-tailored malware, which can adequately be analyzed to better understand the evolutionary state of IoT malware, as well as to build effective detection and classification modules.

2.1.4 IoT Malware Landscape

The public release of the source code of the IoT-based botnet `Mirai` [1] has prompted new actors to easily bootstrap their own botnet and compete with other botmasters over the control of vulnerable IoT devices. Antonakakis et al. [2] have presented the first comprehensive study of the `Mirai` botnet

and described the effect of the shared source code on the release of new specialized variants. Several other works have focused on the customizations of `Mirai` (e.g., `Hajime`, `BrickerBot`) to study the change in their infection behavior [19] as well as to reveal shared password combinations used during brute forcing [39]. Vervier et al. [7] have shed the light on the abruptly changing IoT threat ecosystem, typically dominated by `Mirai`, where new attack players (e.g., Hajime, IoT Reaper) are claiming their share of vulnerable IoT devices by targeting a wider range of firmware vulnerabilities. While these works focus on one botnet, Griffioen et al. [40] have focused on multiple Mirai-like variants competing for the same IoT devices to identify the differences in success between botnets. They exploit 7,500 IoT honeypots and a flaw in the design of Mirai's random number generator to conclude that IoT botnets are not self-sustaining. Alternatively, Torabi et al. [41] leveraged passive network measurements collected from the darknet along with IoT device information to infer compromised IoT devices in the wild.

While previous works hold valuable insights on the threat landscape of IoT malware, further research is needed to prevent the emergence of new intra-family strains and track the evolution of existing ones.

2.2 IoT Malware Analysis

In the past two decades, the security community has focused almost exclusively on fighting generic malware targeting Windows or more recently, Android devices. As a result, a majority of papers have developed techniques tailored to the analysis of PE binaries, the detection of ongoing threats and the prevention of new infection attempts on Windows operating systems. It is only since the appearance of the newsworthy `Mirai` botnet [2] and *Shellshock* [42] that non-Windows malicious software started receiving the same level of attention. In the following sections, we provide background information about the structure of an ELF binary, the difference between statically versus dynamically linked binaries, as well as describe various techniques applied in literature for malware analysis.

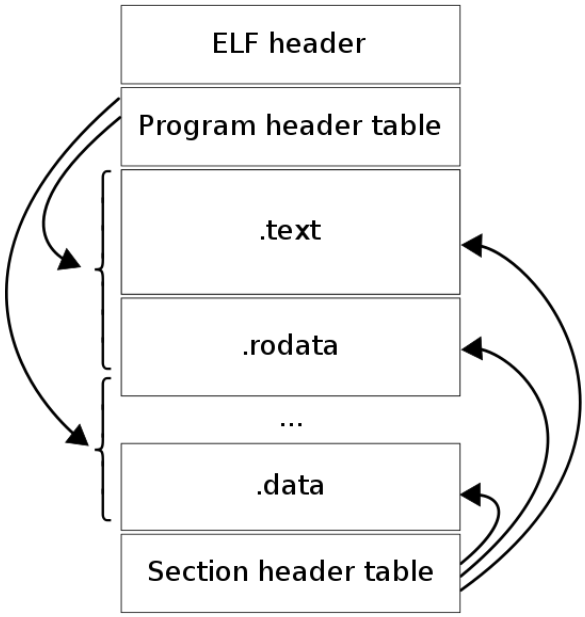


Figure 2.2: Structure of an ELF binary [3].

2.2.1 The 101 of the ELF File Format

ELF which stands for Executable and Linkable Format is a standard Unix blueprint of how machine instructions are stored in an executable code and how they should be interpreted by the operating system. It defines the structure of Linux, Android and BSD executables, libraries, object files, and core dumps. Because of its flexible nature and its support of multiple CPU types and architectures, the ELF file type is adopted by embedded Linux malware. In order to analyze Linux/IoT malware binaries, it is important to understand the ELF file structure, its internal components, and all the information we can extract at our advantage. A set of specific tools and techniques have been developed to aid the dissection and analysis processes of ELF files.

The ELF file format has complicated inner workings that cannot all be explained in a single section in this thesis. Hence, we refer the interested readers to the main ELF and ABI standards [43] for a complete overview. We only focus here on the essential elements that will complement the next chapters.

ELF file anatomy. As shown in Figure 2.2, an ELF file generally consists of the following principal components:

(i) *ELF header.* This part of the ELF file is mandatory. Always located at the beginning, it contains information that ensures that data is correctly interpreted during linking or execution. As illustrated in Table 2.1, using the *readelf* command in Linux, the following information can be extracted from an ELF binary's header: ELF file type, architecture, the entry point to the start of execution, as well as the offsets of the section and program header tables.

(ii) *Program headers.* Also known as segments, the program headers break down the ELF binary to suitable chunks to prepare the executable to be loaded into memory. An ELF binary requires these program headers to run.

(iii) *Section header table.* Sections are used for linking, relocation and debugging purposes. An executable file has four main sections, of which `.text` that contains executable code. This section is loaded only once since its contents will not change and can be extracted using the *objdump* utility in Linux. While segments are essential to create a process, sections on the other side can be omitted. However, they usually offer better granularity to the inspection of a program because they can point reverse engineering tools to the code area (`.text`), to the data variables (`.rodata`), the global offset table (`.got`), among others. Therefore, malicious actors choose to obfuscate the section header table. However, to our advantage, ELF binaries are still highly unobfuscated [6, 44], which allows us to extract the code from the program's text section.

2.2.2 Static versus Dynamic Binaries

Linking plays a major role in the process of building an executable. The source code is usually compiled and turned into machine language instructions. The product of compilation (i.e., object code) is then fed as input to the linker which links the object files together with external libraries to create a functioning program. ELF binaries are divided into two types: statically linked and dynamically linked. A"dynamic" binary depends on external libraries to run properly. Such libraries contain functions related to opening files or creating a network socket. A statically linked binaries on the other hand includes all the library dependencies, which makes it bigger, more portable (i.e.

Table 2.1: Structure of an ELF header.

Field	Description
Magic	ELF type declaration
Class	Architecture (32-bit or 64-bit)
Data	Bit numbering (LSB or MSB)
Version	Object file version
Type	Object file type (e.g., EXEC)
Machine	Expected machine type (e.g., MIPS r3000)
Entry point address	Virtual address where the process starts executing
Start of segments	Offset of the program header table
Start of sections	Offset of the section header table
Flags	Processor-specific flags
Header size	ELF's header size
Segments size	Size of one entry in the program header table
Number of segments	Number of entries in the program header table
Sections size	Size of one entry in the section header table
Number of sections	Number of entries in the section header table
Section header string table index	Section header table index associated with the section name string table

executed correctly on another device without pre-installed dependencies) and harder to reverse engineer (i.e., library functions are difficult to identify). Using the *file* command as shown in Listing 2.1, we can check if a file is statically or dynamically linked.

2.2.3 Malware Analysis Techniques

We present well-established techniques from the security community that we have used in this thesis to study IoT malware.

Static Analysis. Static malware analysis refers to collecting information about a malicious application without running it. Also known as code-based analysis, it is performed by breaking up different parts of the binary file without executing it and investigating each component for maliciousness. For instance, as shown in Listing 2.1, it can can be used to identify each binary's target architecture (e.g., ARM, MIPS), and linking method (static vs dynamic) based on the file headers. Often, the binary file needs to be reverse-engineered using disassemblers (e.g., IDAPro [45], *objdump*) to retrieve the assembly code instructions of the program, control flow graphs (CFGs), or any

```
$ file /bin/fff793a5a0576a896199d56eefed8121

/bin/fff793a5a0576a896199d56eefed8121: ELF 32-bit MSB executable, MIPS, MIPS-I
    version 1 (SYSV), statically linked, stripped
```

Listing 2.1: Example of using the Linux file command to verify whether a binary is statically or dynamically linked.

```
rm -rf %s; pkill -9 %s; killall -9 %s;
cd /tmp || cd /var/run || cd /dev/shm || cd /mnt;
rm -f *; /bin/busybox wget http://AnonIP/bins.sh;
chmod 777 bins.sh; sh bins.sh;
/bin/busybox tftp -r tftp.sh -g AnonIP;
```

Listing 2.2: Example of readable strings extracted from a malware sample with anonymized IP addresses (AnonIP).

embedded strings. In fact, as it is correlated with specific functions and actions inside the program, making sense of the assembly instructions can provide a better visualization of what the malicious program is doing. Moreover, gaining access to the strings, using the *strings* command, can determine whether a sample is packed, but more importantly reveal sensitive information such as the used malicious domains, targeted IP addresses, attack commands, downloaded payloads, etc. Listing 2.2 represents an example of the extracted strings from a malware sample, which is designed to download and execute a malicious file (*bins.sh*) from a possible adversarial C&C server (*http://AnonIP/*). We notice that the malware is trying to eliminate all running processes (e.g., *rm, pkill, killall*) in order to fortify itself, and is trying different commands to download malicious payloads in case one of them fails, as seen in this consequent instruction using the TFTP protocol (e.g., *tftp -r tftp.sh -g AnonIP*). Moreover, *infrastructure analysis* is a type of analysis that allows to filter and identify C&C indicators from the malware strings. Indeed, it is possible to uncover adversarial infrastructure and shared resources, which are used to operate malware-driven cyber attacks. In addition, an analysis of endpoints and their targets reveals useful insights about the underlying dynamics in the IoT malware ecosystem. In fact, we leverage such infrastructre analysis in recent contributions published at the IEEE Networking Letters (2021) [46] and submitted (peer reviewed) at the IEEE TDSC (2021).

In fact, static analysis allows for a quick, scalable and effective analysis of a malicious file without the hassle of executing it. However, it can be evaded relatively easily when malware authors

deploy various obfuscation techniques. To our advantage in this thesis, IoT malware binaries are still widely unobfuscated, which makes static analysis our preferred choice of analysis.

Dynamic Analysis. With dynamic analysis, a malicious file is executed in a controlled environment (e.g., virtual machine) and analyzed to observe its functional and behavioral characteristics. A full system analysis can consist of collecting a malware's system calls, function parameters and network traffic. In fact, dynamic analysis can overcome problems with obfuscation and effectively detect unknown or zero-day threats.

Yet, the dynamic analysis process is time-consuming and malware analysis environments are often recognizable [47], which makes the analysis results susceptible to failure. Hence, it is important to build "life-like" virtual machines that trick a binary into thinking it is running in a real environment. In fact, we built our own IoT-tailored sandbox for malware analysis, as described in Section 2.2.4.

Hybrid Analysis. Hybrid techniques combine both static and dynamic malware analysis techniques to cover each other's shortcomings. Certain actions can be hidden at run-time but may be detected when unpacking the binary file and analyzing its assembly code. Despite their benefits, the implementation of such in-depth analysis techniques requires time, expertise and a thorough analysis of the malware, which is costly and unscalable.

AI-assisted Malware Analysis. AI- and ML-based cybersecurity offers data-driven processes that can enable security systems to efficiently identify and respond to malware in real time. For instance, Machine and Deep learning models have been proposed to perform static malware analysis by extracting features from multi-modal views of the malware (e.g, image-based, strings-based features) [10, 48, 49]. Such next-generation analysis techniques offer several benefits to scalability and efficiency. Additionally, deep learning methods do not require expensive and bias feature engineering/extraction, compared to machine learning-based models. As such, AI-based techniques offer a number of benefits, but they still require costly training and testing to adapt to the changing threat landscape, require labeled data in the case of supervised learning, are exposed to overfitting and underfitting, and might be hampered by adversarial examples and detection evasion techniques.

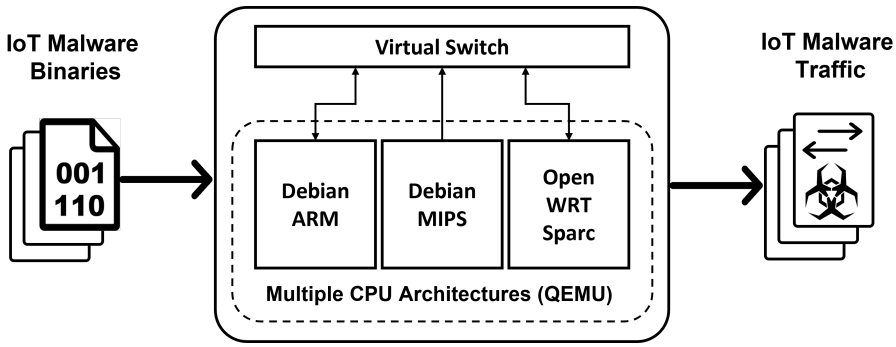


Figure 2.3: An overview of the multi-architecture dynamic malware analysis pipeline.

2.2.4 IoT Malware Sandboxing Environment

IoT malware employ capabilities to target specific types of IoT devices and perform different malicious operations. As a result, malware is highly specialized and thus requires a comprehensive emulation and dynamic analysis platform to explore its behavior. In this thesis, we decide to complement our statically extracted features by dynamically analyzing the communication behavior of our malware samples and profiling it at the operating level. Hence, we build architecture-specific virtual machines using the `QEMU` emulator to execute each sample and collect their network traffic. Figure 2.3 represents our virtual cross-compilation dynamic malware analysis environment with bridge networking. We run malware binaries for a period of 30 minutes to capture the generated network packets at the gateway using `TShark`. We found that about 95% of malware will engage in communication with their C&C that either block or loop infinitely. In fact, we were only able to collect useful traffic information from 0.5% of analyzed samples. These findings reflect the challenges, documented by earlier works [8, 50], associated with dynamically executing and collecting IoT malware binaries' network traffic communication. Complementary techniques and customizations (e.g., impersonating C&C server) are required to ensure proper execution of device-specific IoT malware in a controlled environment. Hence, addressing the limitations of large-scale dynamic analysis of IoT malware is out of scope of this thesis.

2.2.5 Leveraged Dataset

In this thesis, we leveraged a number of resources to obtain a representative dataset for further empirical data analysis and experimentation purposes. We utilized well-known online malware

Table 2.2: Distribution of malware by family.

Label	Count (%)
Mirai	40,974 (55.05)
Gafgyt	3,976 (5.34)
Tsunami	956 (1.28)
Dofloo	464 (0.62)
Others	122 (0.16)
Unknown	2,664 (2.23)
Unseen	24,271 (32.60)
Total	74,429 (100)

repositories such as VirusShare [51] and VirusTotal [9] along with a specialized IoT honeypot (IoT-POT [6]) to obtain over 90,000 IoT malware samples that were detected between 2018 and 2021. For consistency purposes, we performed pre-processing steps to filter out corrupted or non-executable files (e.g., HTML/ASCII files), ending up with 74,429 IoT malware binaries. To label these samples, we have retrieved their VirusTotal (VT) analysis reports and processed them with AVClass [52], which determines the most likely family name attributed to malware samples by applying a majority rule on reported labels from multiple anti-virus engines. Table 2.2 reports the top identified families, with `Mirai` and `Gafgyt` dominating the dataset. Such imbalance in the data across different families is a mere reflection of the monopoly inflicted by `Mirai` and its descendants on the IoT threat landscape. In fact, the effectiveness of the `Mirai` family motivates adversaries to reuse/recycle the `Mirai` source code, with most of the "new" IoT botnets to represent mere modifications of the `Mirai` code base. Moreover, the analysis of Internet-scale scanning activities generated by infected IoT devices confirms the prevalence of `Mirai`-like malware in the wild [40, 41, 53].

Missing Classes. It is worth noting that for 34% of the collected samples, AVClass [52] failed to reach a consensus for a common family name, as 2,664 of them were not associated with known IoT malware families to anti-virus engines (*Unknown*), and 24,271 were never found in VirusTotal reports (*Unseen*). This is an indication that the identified malware binaries can be either new, or have not been detected yet by anti-virus vendors. In fact, it is unrealistic to assume that security analysts are aware of all malware families deployed in the wild. Yet, it stands to confirm the effectiveness of

Table 2.3: Inconsistent labels assigned to the same variant by 12 out of 62 independent AV engines on VirusTotal.

File (md5)	*ff8c3145d910221c6c6168cca0cd85fd*
Engine	**Given Label**
Ad-Aware	Trojan.Linux.Agent.DML
AhnLab-V3	Linux/Mirai.Gen2
Antiy-AVL	Trojan[Backdoor]/Linux.Mirai.b
Arcabit	Trojan.Linux.Agent.DML
Avast	ELF:Mirai-A [Trj]
Avast-Mobile	ELF:Mirai-DN [Trj]
BitDefender	Trojan.Linux.Agent.DML
ClamAV	Unix.Trojan.Mirai-7100807-0
Cynet	Malicious (score: 85)
Cyren	E32/Mirai.G.gen!Camelot
DrWeb	Linux.Mirai.4934
Emsisoft	Trojan.Linux.Agent.DML (B)

honeypots towards a promptly collection of IoT malware samples.

Coarse-grained Labels. Another common oversight is that the labels assigned by AV vendors are often inconsistent and coarse-grained, and therefore unable to capture the code reuse between IoT malware and their evolutionary characteristics. For instance, it is still unclear how many variants of the Mirai botnet have been observed in the wild. Table 2.3 describes the inconsistent labels assigned to the same variant by 12 out of 62 independent AV engines. In fact, 7 engines labeled the file as a generic Linux Trojan, while the rest assigned incompatible labels such as Mirai.Gen2, Mirai-A, Mirai-DN, Mirai.4934, Mirai.G, Mirai-7100807-0. The diversity of the AV market and the lack of standards for transparent malware family labeling creates a lot of disorganisation and uncertainties when it comes to identifying a representative and certain malware variant name.

Malware Detection Timeline. To verify that our malware labels are accurate and reliable, we considered a 5 months time period between the end of our malware data collection and our performed malware family labelling to avoid collecting incomplete or inaccurate malware family labels. Specifically, at the time when this research was conducted, the Unseen identified unknown malware samples were never seen on VirusTotal reports even after 5 months from being detected by

IoTPOT. This affirms the effectiveness and ability of specialized IoT honeypots to promptly detect various IoT-tailored malware.

Note that all the collected IoT malware data is available for research purposes and can be directly requested from the above-named sources (e.g., IoTPOT [34]). Unfortunately, the restricted sharing policies instilled by the data providers prevents us from directly sharing the analyzed data with the research community.

2.3 IoT Malware Detection and Classification

Several works have devised ML/DL methods that leverage a combination of features from different malware characteristics for IoT malware detection and classification.

Grayscale Images. An original way to represent an executable file is to reorganize its byte code as a grayscale image, like [54] where every byte was interpreted as one pixel in the image. Considering such malware representation, Ahmadi et al. [55] extracted a set of features from the grayscale image, such as Haralick features and Local Binary Pattern features, and achieved an accuracy of 96.90% and 97.24% respectively using the XGBoost classifier. Beppler et al. [56] evaluated and compared global (GIST) and local (LBP) descriptors using a multitude of classifiers (e.g., KNN, SVM, DT, RF, CNN). Convolutional Neural networks were also used for classification of malware represented as images. Gibert at al. [57] developed a deep learning system based on a CNN that learns visual features from executable files to classify Windows malware into families. Su et al. [58] proposed a lightweight solution for detecting and classifying IoT DDoS malware and benign application on IoT devices using a small size convolutional neural network. They achieved 94% accuracy, however their used dataset (500 samples) is very limited in size and diversity, and their considered image size (64x64) is attributed empirically with no consent about the best one. A more prudent resizing of 128x128 has been shown to produce lower variation and maintain a high accuracy rate in all cases [56].

Malware Strings. Extracted malware strings can provide useful indicators associated with a suspect binary and its functionalities. Several works [59–61] have adopted the use of string information as a feature vector for malware classification and signature generation. For instance, Tian

et al. [59] leveraged printable strings extracted from Trojans and viruses to perform classification, and evaluated their approach on a multitude of classifiers (e.g. SVM, RF, Instance Based 1 (IB1), Adaboost), and showed that IB1 and RF classification methods were the most effective. Alhanahnah et al. [60] leveraged N-Gram strings-analysis for correlating and clustering malware samples based on their strings similarities. In addition, Nguyen et al. [61] proposed a novel approach for Linux IoT botnet detection based on the combination of Printable String Information (PSI) graph and CNN classifier. Their evaluation results show that PSI graph CNN classifier achieves an accuracy of 92%. Still, to build PSI Graphs, their approach required generating malware Control Flow Graphs (CFGs), which is a complex task that requires time and domain knowledge. Nevertheless, to the best of our knowledge, no previous work has treated malware strings as a text classification problem and leveraged the use of end-to-end learning and NLP techniques for the classification of IoT malware using such information.

Multimodal Learning. While these approaches use one representation of the data to extract features that are used for malware classification, in practice, these single-level features might not always be available for analysis (e.g., due to obfuscation). Thus, efforts are being put to design models that leverage multiple data modalities from different malware characteristics. Some approaches like [55] rely on fusing multiple hand-engineered features (e.g., frequency of opcodes, image representation, entropy statistics, etc.) into a single feature vector that is used as input to a traditional ML algorithm. On the other hand, other researchers leverage an ensemble of individual classifiers that process a different modality of data to precisely classify malware [48, 62–64]. Alhanahnah et al. [60] leveraged the benefits of a multi-level approach for malware clustering and signature generation to detect cross-architecture IoT malware using features such as code statistics feature, high-level structural similarity, and N-gram string features. Gibert et al. [48] leveraged the use of multiple features from different modalities of data, combined with deep learning algorithms to detect/classify Windows malware. Despite the fact that their approach produced high classification accuracy, their implemented classifier relies on features (e.g., API function calls, assembly language instructions) that require rather sophisticated reverse-engineering techniques with deep learning network models, which tend to be resource consuming. Yet, in this thesis, while we deal with certain limitations in the context of IoT, we leverage features that can be retrieved without

the need to perform expensive pre-processing and feature-engineering tasks. In addition, while our classification approach produces improved accuracy (99.78) as presented in Section 3.6, our implemented DL models can qualify as a lightweight solution for enhancing the security of the IoT paradigm through effective malware classification and threat mitigation.

Transfer Learning. Zhao et al. [65] proposed a malware detection method of code texture visualization based on an improved RCNN combining transfer learning, which achieves an accuracy of 92.8%. Bendiab et al. [66] proposed an IoT malware traffic analysis approach using deep learning and visual representation for fast detection and classification of new malware. They evaluate their proposed method on a dataset of 1000 pcap files of normal and malware traffic and achieve 94.5% accuracy rate for detection using ResNet50. Both of the above mentioned works [65, 66] rely on deep neural networks with hundreds of layers, such as ResNet50 and ImageNet, whose heavy computational cost of multiple layers can be difficult to handle by resource-constrained IoT devices. In contrary, our multi-level IoT malware classification approach, which achieves an overall accuracy of 99.78, relies on lightweight CNN and LSTM models that transfer their learned features and weights throughout the proposed architecture, respectively. Moreover, Bendiab et al. [66] rely on a very small number of pcaps which can hinder the generalizability of their approach and their classification results. In addition, relying on IoT malware network traffic has its limitations. It is difficult to configure a dynamic malware analysis environment that meets the requirements of IoT executables to function correctly and the collection of a large number of IoT malware network traffic pcap files, where the malware is actually communicating with its C&C server and revealing its behavior, is still challenging [8, 50].

Other features. Recent works have applied deep learning methods on more complex malware representations such as control flow graphs. Yan et al. [67] used machine learning techniques to classify malware programs represented as their control flow graphs. Their MAGIC framework achieves high accuracy (99.25%). Nevertheless, their approach is shown to be coarse to detect the malicious programs with a high false negative rate. For instance, some DDoS samples or worms may share the same graph structure as benign software. Our model addresses this by preventing the co-adaptation of the subnetworks to a specific feature type. Therefore, even if two different binaries may share the same characteristics from one modality, our classifier would still achieve

good performance by learning distinctive feature from the other data modality, hence resulting in less false negative rate. Alasmary et al. [68] proposed an adversarial machine learning detection system for IoT malware based on control flow graph feature representations. Both [67, 68] chose to extract complex and time-consuming features for their analysis, while we leveraged the coupled nature of IoT malware [44], their general lack of obfuscation [6, 44], their lack of diversity [41, 53] and therefore, the ability of deep learning methods to automatically extract a set of descriptive static features from their images and strings without relying on feature-engineering and domain's knowledge. We show that our multi-fusion approach (§3.3) is feasible, accurate and performs well on the used image- and strings-based features in the context of IoT, as shown in Sections 3.6.1 and 3.6.2. Yet, Alasmary's approach [68] is robust against Adversarial Examples (AEs) and eliminates the model's vulnerability to AEs. We consider complementing in the future our effective multi-level classifier with an AEs detection component to make it robust against adversarial attacks (see Section 3.7.2).

2.4 Malware Evolution and Lineage Inference

Malware is constantly evolving to adapt to survival needs, bug fixes, and feature additions. Lineage studies are most useful when applied to malware as version information is usually not available [69, 70]. The first empirical attempt to reconstruct a digital phylogeny dates back to 1995 when Hull et al [71] studied the *Stoned* computer virus. Inspired by the evolution of species and molecules, Goldberg et al. [72] and iLINE [69] produced malware phylogeny trees using a directed acyclic graph (DAG). Dumitras et al. [73] studied malware evolution to find new variants of well-known malware and provided a general blueprint for addressing malware lineage using a combination of static and dynamic features, and time-related information. Karim et al. [74] reconstructed phylogeny trees using a code fragment permutation-based technique to understand how new malware is related to previously seen malware. Lindorfer et al. [75] investigated the malware evolution process by mapping API calls to disassembled code in order to identify mutations in the malware family. Calleja et al. [76] identified code reuse between benign software and different Windows malware families observed over a period of 40+ years. Their observations ranged from common

utility functions to anti-detection routines to credentials for brute-forcing attacks.

Binary Code Similarity. Moreover, several techniques for binary similarity gained momentum as they can be applied for malware lineage inference. In a recent survey, Haq et al. [70] highlighted the strengths and weaknesses of 61 approaches on binary code similarity, including those used for malware evolution [69, 75, 77, 78]. BEAGLE was proposed by Lindorfer et al. [75] to study malware evolution by comparing binary code in terms of API calls extracted using behavioral analysis. Huang et al. [77] identified code reuse in two Windows malware families by computing the similarity between functions extracted from binaries at the instruction, basic block and CFG levels, while Jang et al. [69] have combined low-level binary features, code-level basic blocks and binary execution traces for lineage construction. Khoo et al. [79] proposed a matching approach based on n-grams computed on instruction mnemonics and graphlets. Existing solutions have been designed around computing CFGs and matching procedures that cannot be adapted to compute a constant size signature of a binary on which a similarity measure can be applied. They also cannot be immediately extended to cross-platform similarity and therefore cannot be applied on Linux-based IoT malware. Cozzi et al. [44] took this opportunity to identify code similarities between IoT malware families using function-level binary diffing. However, they resorted to popular off-the-shelf binary diffing tools not tailored for IoT, which required a substantial amount of manual adjustments and validation.

Works based on *embeddings*. Recent advancements in machine learning techniques have seen effective applications in binary code similarity. Ding et al. [80] have recently proposed an assembly clone search approach named *Asm2Vec*, which learns a vector representation of the sequence of instructions executed on a certain path of the CFG. While it outperformed several state of-the-art solutions in the field of binary similarity, Asm2Vec is not directly applicable for semantic clones across architecture as it only generates single-platform embeddings, and it has the performance overhead of performing random walks on CFGs. In another work, Xu et al. [81] proposed *Gemini*, a neural network-based approach to compute binary function embeddings based on an *annotated CFG*, a graph containing manually selected features. However, their annotation approach based on manual feature selection can introduce human bias, by preferring, for instance, arithmetic instructions over others. In *InnerEye* [82], Zuo et al. apply the idea of Neural Machine Translation (NMT) to find

similar CFG blocks. However, it is not clear how such embeddings are a representation of entire functions. Secondly, the used LSTM architecture is burdened by long term dependencies/memorization, hence does not cope well with long sequences of instructions [83]. *DeepBinDiff* [84] has been proposed to find differences between two binaries by leveraging deep neural networks and greedy graph matching. Yet, DeepBinDiff requires call symbols and strings which would be stripped away or obfuscated in statically linked malware, hence it is a single architecture solution.

2.5 Concept Drift in Machine Learning-based Security Applications

In the context of data-driven machine learning applications, the dataset used for training a model plays a major role as its properties define the model's behavior. In many situations, it is assumed that the distribution of the data is stationary (i.e., not changing over time), specially when the same data used to train a model is later used during production. However, the environments in which the models are deployed are usually dynamically changing over time. Such changes can include, but are not limited to, attackers constantly modifying their attack vectors. As a result, this change in the data distribution of a machine learning model, called *concept drift* [85], makes it difficult to generalize existing learning models that were trained with older data to new, previously-unseen behaviors.

Being a widespread problem in the security community, detecting concept drift is critical in many real-world security applications, such as anomaly detection, fraud detection, malware classification, or intrusion detection. To address it, Gama et al. [86] proposed a performance-based statistical technique aimed at tracking changes in the error rate of a model. Bifet et al. [87] presented `ADWIN`, a new algorithm for retraining according to the rate of change observed from the data in a specific time window. However, both these drift detectors operate in a supervised environment and require access to labelled ground truth data for retraining which are difficult to obtain in security applications. More importantly, periodic retraining requires knowing *when* the model should be retrained which is difficult, and delayed retraining can leave an outdated model vulnerable to new attacks. Other related works have relied on the prediction decisions of a learning model as a by-product of the classification process [10, 88, 89]. However, it is likely that a drifting data sample

(e.g., malware variant) that does not belong to any class will be assigned with high confidence to the wrong class (i.e., closed-world assumption). More recent works [90, 91] have mitigated such bias by computing a *non-conformity* probabilistic measure between the new sample and each of the existing classes to determine its fitness in each class. Although useful, these approaches cannot draw concrete conclusion on drifting (evolving) samples and lose their effectiveness on high dimensional data. Pendlebury et al. [92] present `Tesseract` to identify temporal and spatial bias associated with incorrect training splits and unrealistic assumptions in dataset distribution, as well as quantify the impact of errors on classifier performance. They particularly focus on Android malware and evaluate two well-known Android malware classifiers, `DREBIN` [93] and `MAMADROID` [94] against concept drift. We follow their methodology to observe our classifier's prediction qualities against evolving `Mirai` samples in Section 4.5.2. Besides, the authors of `CADE` [95] have recently attempted to develop an unsupervised concept drift malware detection method by using neural networks. They specifically used an auto-encoder coupled with a contrastive loss to compress the data and learn an effective distance measure between samples of different classes. While their resulting distance function can efficiently detect and rank drifting malware samples from distinct classes, their approach is not tailored to in-class drifting samples, which is more relevant in the context of IoT.

Chapter 3

A Multi-Dimensional Deep Learning Framework for IoT Malware Classification and Family Attribution

3.1 Overview

Internet of Things (IoT) devices have been integrated in different aspects of our everyday activities. These Internet-connected devices are mainly used to improve user experiences by facilitating information sharing, monitoring, and communication. Despite their benefits, the rising number of IoT-tailored malware, which aim at utilizing compromised IoT devices (e.g., weak authentication) towards coordinating large-scale cyber attacks, has posed a major threat to the overall Internet ecosystem [2, 96]. For instance, the `Mirai` botnet was leveraged in the famous cyber attack on Dyn (major US DNS service provider) in October 2016 [2], resulting in one of the largest recorded DDoS attacks on the Internet. More importantly, the release of the `Mirai` source code fueled the rapid evolution of more advanced and sophisticated `Mirai`-like malware such as `Hajime` [12], `Satori` [13],

The work has been published in: IEEE Transactions on Network and Service Management (Volume: 18, Issue: 2, June 2021) [10].

and `BrickerBot` [14], to name a few.

It is imperative to evaluate the security of the IoT paradigm as well as develop rigorous mitigation approaches against the spread of IoT malware. These tasks are challenging in the context of IoT due to the lack of empirical data about existing IoT malware and the lack of knowledge about the behavioral characteristics of malware-infected IoT devices. To overcome these challenges, a number of IoT specialized honeypots have been deployed to obtain detailed information about existing IoT malware, including the malware executable/binary [7, 34].

Additionally, static malware analysis techniques can be used to build a better understanding about IoT malware, while extracting features that can improve attack mitigation by developing efficient malware classification techniques using machine/deep learning algorithms. For instance, a number of strings-based features have been utilized to devise ML/DL methods for malware classification/clustering [59–61]. Moreover, image-based techniques, which extract features from the image representation of malware binaries, have been effectively used in different contexts [54–58]. Finally, models that leverage a combination of features (e.g., CFGs, statistical features, etc.) from different malware characteristics have been proposed for malware classification [48, 55, 60, 62–64].

To this end, we propose a multi-level approach that leverages a combination of static features along with DL techniques for effective IoT malware classification and family attribution. Our objective is threefold: (i) to leverage IoT-specific properties such as the lack of widely deployed sophisticated malware obfuscation [6, 44] for extracting static features from different representation of the IoT malware binaries that are not empirically feasible in other contexts (e.g., widely obfuscated Windows PE or Android malware) [97], (ii) to leverage deep learning methods capabilities to automatically extract static features without relying on expensive feature-engineering, and (iii) improving the overall classification accuracy by building a multi-dimensional DL architecture that utilizes different representations of the target IoT malware binaries.

To achieve our objectives, we leverage about 70,000 real instances of IoT malware binaries/executables obtained from VirusTotal [9], VirusShare [51], and a specialized IoT honeypot (IoTPOT [8]) over a period of 20 months (September 2018–May 2020). We utilize AVClass [52] to investigate IoT malware family labels as perceived from VirusTotal reports while identifying about

26,000 IoT malware samples that were "unknown" or "unseen" by major antivirus vendors. Additionally, motivated by the lack of malware obfuscation in the IoT context, we devise string- and image-based analysis techniques using convolutional neural network (CNN) and long short-term memory recurrent neural network (LSTM), respectively. Consequently, we implement our multi-level DL architecture by fusing the learned features from each sub-component through a neural network classifier. Finally, we perform a series of experiments using about 10,000 IoT malware samples from four different predominant families and evaluate the effectiveness of our approach in comparison to state-of-the-art approaches that implement single-level strings-based and/or image-based classifiers.

3.2 Contributions

To this end, this work makes the following main contributions:

- Given the challenges associated with IoT malware classification and family attribution, in this paper, we are among the first to introduce a holistic, multi-level approach for analyzing IoT malware by combining the benefits of static malware analysis with deep learning classification techniques. More importantly, despite the fact that IoT malware families tend to be similar in terms of implementation and overall behaviors [44], our results show that the proposed multi-level approach can be used to perform effective and efficient classification by considering granular characteristics from the analyzed malware binaries.

- We implement the proposed approach by utilizing DL methods to automatically extract static-based features to overcome the challenges associated with feature-engineering methods. Moreover, we evaluate the multi-level deep learning model with 10,234 recently collected IoT malware executable binaries. The results indicate a significantly improved classification accuracy (accuracy=99.78% and F1-score=99.57%), as compared to classifiers that rely on a single modality of data.

- To the best of our knowledge, we are among the first to obtain and analyze a large and representative sample of real IoT malware executables, which contains a variety of IoT malware

variants detected in recent years. While our analysis results indicate the effectiveness of the proposed classification approach for attributing malware samples to known families, we also leverage the multi-level classifier to predict the labels of 24,271 unknown malware samples that have not been detected/labeled by major AV vendors. Moreover, while our extended strings-based similarity analysis corroborates the labeling outcomes, we uncover indications of new `Mirai` variants related to the `Covid-19` pandemic, which highlights the rapid evolution of IoT malware found in the wild.

3.3 Multimodal Deep Learning Framework

In this work, we propose a multi-dimensional DL malware classification approach that can detect and attribute malware executable binaries to known IoT malware families. Our aim is to develop and evaluate a classification approach that automatically learns static features from multiple representation of the malware binary, while improving the overall classification accuracy through combining multiple modalities. In particular, we attempt to answer the following research questions (RQs):

(1) *How can we utilize static malware analysis techniques to develop an effective multi-level classifier for IoT malware family attribution? Does a multi-level deep-learning approach that combines learned characteristics of malware from various data representation yield a higher classification performance compared to single modality DL algorithms?*

(2) *How can we benefit from next-generation malware analysis techniques to overcome the challenges associated with feature-engineering? How effective is the proposed multi-level deep learning approach as compared to state-of-the-art ML approaches that utilize various combinations of features and feature engineering techniques?*

(3) *How to leverage the developed multi-level classifier to detect new or unknown malware samples given the information about existing malware families?*

The architecture of our multi-level deep learning framework for IoT malware classification is

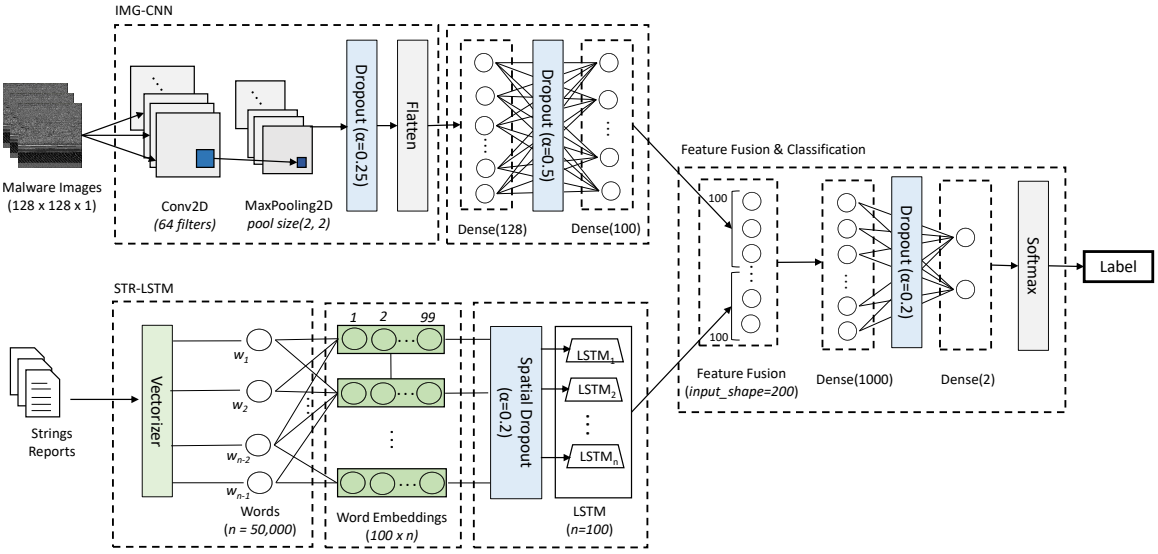


Figure 3.1: Overview of the Multi-Level Deep Learning Malware Classification System.

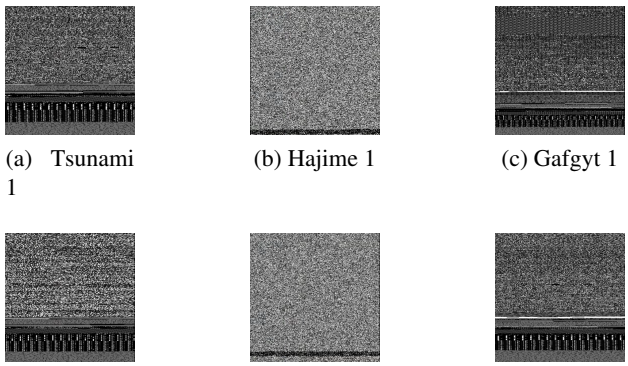
shown in Figure 3.1. In the proposed framework, the input is an ELF executable binary for Linux-based systems, while the classification outcome represent the IoT malware family label. The classification module consists of 3 main components: (1) image-based component, (2) string-based component, and (3) the feature fusion and classification component. The strings- and image-based components extract/learn corresponding features from different representation of the malware. Consequently, the final component is responsible for fusing the learned features into a shared representation, which is used to produce the final classification outcome. An in-depth analysis of the different sub-components and features types chosen, is provided in the next sections.

Leveraged Dataset. In this work, our dataset consists of a total of 74,429 IoT malware binaries collected between *2018-09-14* and *2020-05-25*, representing 18 different malware families labelled by using AVClass [52] and VirusTotal [9]. More information about the collection and labelling process, and the data cleaning steps is provided in the Background Section 2.2.5.

3.4 Feature Modalities

3.4.1 Image-based Component

In this work, we use the approach proposed by Nataraj et al. [54] to visualize malware binary files as grayscale images. In particular, a malware binary can be read byte-by-byte as a vector of 8



(a) Tsunami 1 (b) Hajime 1 (c) Gafgyt 1

(d) Tsunami 2 (e) Hajime 2 (f) Gafgyt 2

Figure 3.2: Grayscale images of malware samples belonging to the `Tsunami`, `Hajime` and `Gafgyt` Families.

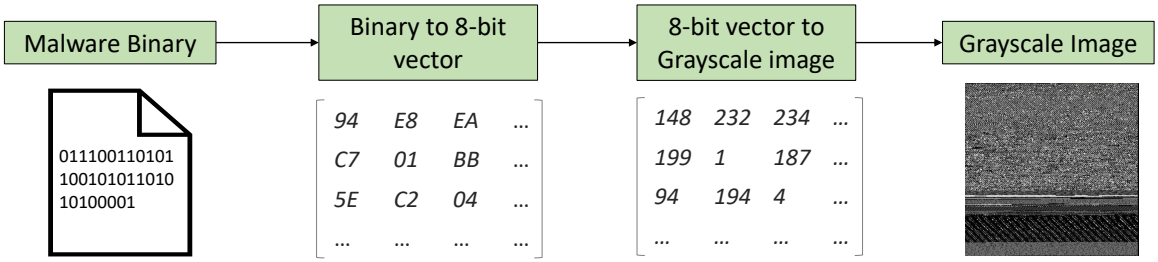


Figure 3.3: Process of visualizing a malware as a grayscale image.

bit unsigned integers and then organized into a 2D array (Figure 3.3). This can be visualized as a gray scale image whose pixel values range from 0 to 255 (0: black, 255: white). As illustrated in Figures 3.2(a–f), we can clearly observe the high resemblance between the image representation of two different malware samples that belong to the same malware family, respectively.

The literature has demonstrated the effectiveness of malware classification techniques using malware image representation. Therefore, as depicted in Figure 3.1, the image-based component takes the bytes representation of a malware grayscale image as input. In our specific implementation, we adopt CNN and LSTM neural networks for our image-based classification algorithms. Additionally, we perform hyper-parameter tuning to select the best combinations of hyper-parameters. We evaluate the performance of both algorithms and present the optimal architecture, which was selected for our final multi-level model in Section 3.6.1, respectively.

3.4.2 String-based Component

In addition to the image representation, we utilize reverse-engineering techniques to extract meaningful strings from the binary code. Specifically, we utilize the *strings* utility in Linux to extract printable strings with three or more bytes. Additionally, we are interested in identifying strings that exhibit similar contextual information, which can help towards grouping IoT malware samples according to the occurrences of the same strings. Such embedded strings can provide clues about the suspect malware and its functionalities (e.g., attack commands, IP addresses, filenames, unique strings, etc.). A detailed example of extracted strings showing an analyzed malware trying to download and execute a malicious file from a possible adversarial C&C server, is illustrated in Listing 2.2 in the Background Section 2.2.3.

Given the extracted strings files, we utilize natural language processing (NLP) techniques to tokenize the top 50,000 most common words in each file. A naive approach to convert words to vectors is to assign each word with a “one-hot vector", which means that the vector would be all zeros except one unique index for each word. However, this type of word representation can introduce substantial data sparsity. Instead, we adopt a continuous vector space representation of the extracted words (embeddings) that allows semantically similar words to be mapped to nearby points, thus encoding useful information about the words' actual meaning/use in the text. Word embeddings are usually joint with neural network models for document classification. Accordingly, we implement CNN and LSTM models for our text/strings classification. Consequently, we perform 10-fold cross validation to evaluate/compare the effectiveness of the implemented models. The optimal architecture, which was configured following a grid search over the hyper-parameters of the networks, is presented in Section 3.6.2.

Malware Obfuscation. It is a common technique deployed by malware writers to hide all or parts of their implementation while avoiding rapid analysis and detection by conventional static and signature-based techniques [98]. We leverage FLOSS [99] tool to investigate the presence of obfuscated malware strings in the analyzed samples. Interestingly, we found that the obfuscated IoT samples were mainly packed by off-the-shelf tools such as UPX [100], which can be easily de-obfuscated. Indeed, our analysis confirms the lack of sophisticated malware obfuscation in IoT by

extracting and de-obfuscating strings from the majority (about 76%) of the IoT binaries. To maintain consistency, we decide to use malware samples that were successfully de-obfuscated throughout our analysis, while discarding the remaining samples from further analysis. Note that although our proposed approach does not consider obfuscated malware binaries, it is still effective in the context of IoT, where it can be leveraged to analyze a significant portion of the detected IoT malware samples in the wild.

3.5 Fusion Component and Classification

As illustrated in Figure 3.1, each sub-component in our framework extracts features from a different representation of malware, i.e., a different data modality. The fusion component is responsible for combining the learned features from multiple sub-component into a shared representation, which is used to enable final classification outcomes. It this work, we aim at demonstrating the effectiveness of applying intelligent fusion of different modalities of features to achieve better classification outcomes, as compared to using single-level classifiers.

To achieve this, we perform per-component pre-training before the final feature fusion and classification. This step is done to avoid overfitting a subset of features that belong to one data modality over the others [48]. Additionally, we pre-trained each component separately while optimizing their hyper-parameters to initialize each component in the multimodal neural network with the optimal learned pre-trained weights, respectively. This is an idea borrowed from transfer learning and feature fusion, where the knowledge and the features learned by each model are transferred into the multimodal neural network to save training time, while converging faster towards better classification results [48, 101]. During the process of transferring knowledge, the following important questions must be answered:

What to transfer & when to transfer: We must comprehend which part of the knowledge learned can be transferred from the source to the target in order to improve the performance of the target task. We should aim at utilizing feature fusion to improve the target classification results and not degrade them. For that reason, we must first pre-train our sub-component deep learning models and record their performance, and then proceed towards fusing the most relevant learned features

by each sub-component to compare the classification results with the multi-level model.

How to transfer: Once the first two questions above have been answered, we can proceed towards identifying ways of transferring the knowledge across different data modalities. Deep learning models are layered architectures that learn various features at different layers. All the layers are finally connected to a *fully connected* layer that is responsible for generating the final output. The key idea of transferring knowledge is to leverage the pre-trained models' weighted layers to extract features, and abandon the models' final classification layer. Hence, each sub-component will act as a feature extractor for the final multi-level deep learning model, which will fuse the different features learned into a joint multi-modal representation.

Feature fusion and classification. As shown in Figure 3.1, the learned representations I of the image-based component and S of the string-based component are continually generated across multiple fully connected layers during the training phase. Consequently, the features learned by the last fully connected layer of each sub-component are fused into a shared multi-modal representation at the final phase. Note that vector I of size i and vector S of size s are fused into a vector M of size m, where $m = i + s$. This joint multi-modal representation M is then fed into a neural network with two fully connected layers and a dropout layer. The last fully connected layer is responsible for classifying a malicious binary as follows: $prediction = softmax(b_c + W_c P)$, where prediction is a vector of size C (number of classes), and W_c and b_c are the weights and biases of the layer. The *softmax* function outputs the probability of a malware to belong to any of the malware families in the training set. The sizes of vectors I and S are determined during the configuration of the network. Hyperparameter optimization is performed accordingly to set the numbers of hidden units for yielding the best results (see Section 3.6). It is important to realise that combining a larger number of features in the final layer will always result in a higher dimensional feature set, which will negatively affect the overall classification outcomes.

Model Evaluation. To compare the effectiveness of the deployed models, we rely on standard machine learning measures such as accuracy, precision, recall, and F1-score. Precision is the ratio of correctly classified IoT malware samples over all the IoT malware samples designated as such ($precision = \frac{t_p}{t_p + f_p}$). The recall is the ratio of correctly classified IoT malware samples over the total number actually existing in the test data ($recall = \frac{t_p}{t_p + f_n}$). While the precision allows the model to

designate only the actual relevant samples as relevant, the recall validates the model's ability to find all relevant samples within a given dataset. The F1-score combines these two metrics together by taking the weighted average (i.e., the harmonic mean) of precision and recall ($F1 = 2.\frac{precision.recall}{precision+recall}$). Finally, we consider the best model according to the macro F1-score, because the accuracy measure by itself might be misleading when used with imbalanced data, whereas the macro F1-score metric gives more importance to False Negatives and False Positives.

3.6 Experimental Results

In this section, we use empirical data to evaluate the effectiveness of the implemented classification approach, while comparing its outcomes to the single-level modality approaches and the state-of-the-art ML techniques with feature engineering. To develop our proposed multi-level malware classification model, we use Keras API to implement the corresponding image- and strings-based components using both CNN and LSTM models. The objective is to evaluate the effectiveness of two different implementation of each components using CNN and LSTM, while choosing the final model that produces the best accuracy and F1-score outcomes.

Additionally, to address the problem of class imbalance within the training dataset, we apply data resampling to obtain 10,234 malware samples representing four prominent IoT malware families: `Mirai` (5,927), `Gafgyt` (3,227), `Tsunami` (776), and `Dofloo` (304). Moreover, we scale the calculated loss for each observation in the models by the appropriate class weight to assign more significance to the losses associated with the minority classes [102, 103]. To validate the stability and generalizability of the deployed models to an independent (unknown) dataset, we leverage a stratified version of k-fold cross validation ($k = 10$). We calculate the average model score across multiple validation iterations while preserving the class distribution in the train and test sets for each evaluation of a given model. Accordingly, the dataset, which consists of 10,234 malware samples, is divided into k subsets, where the models are trained with $k-1$ subsets and tested with the last subset over k iterations. Further, to select the best model implementation, we perform grid search using Talos [104], which automates the hyper-parameters tuning and model evaluation processes. The optimized hyper-parameters are presented in Table 3.1.

Table 3.1: Hyper-parameter tuning/selection and 10-fold cross validation performance evaluation results for the selected DL models.

Parameters	**Space**	**IMG-CNN**	**STR-CNN**	**IMG-LSTM**	**STR-LSTM**	**Our Model**
Num. of filters (f)	32,64	64	32	-	-	-
Num. of units (u)	2,50,100,128,1000	128, 100	-	128	100	1000, 4
Kernel size (w x w)	(2,2),(3,3),(4,4)	(3,3)	(4,4)	-	-	-
Pool size (p)	2,3	2	2	-	-	-
Batch size	32,64,128	64	64	64	64	64
Epochs	10,15,20,30	15	10	10	10	10
Activations	Relu, Elu	Relu	Relu	Relu	-	Relu
Dropout	(0, 0.2, 0.25, 0.3)	0.25	0.5	0.2	0.2	0.3
Spatial droput	(0, 0.1, 0.2)	-	-	-	0.2	-
Validation split	(0.1, 0.2, 0.25, 0.3)	0.2	0.2	0.25	0.2	0.2
Accuracy	–	0.9722	0.9886	0.9711	0.9840	**0.9978**
Macro F1 score	–	0.9721	0.9851	0.9702	0.9820	**0.9957**

3.6.1 Evaluating the Image-based Component

We present the analysis of the performance of two deep learning algorithms, CNN and LSTM, that we designed to classify IoT malware based on their bytes-based image representation. As shown in Table 3.1, the results of the 10-fold cross validation on the implemented models illustrate that both models perform significantly well with high accuracy and F1-score outcomes. However, the CNN implementation of the image-based components yields slightly better outcomes, with about 97.2% for both the classification accuracy and macro F1-score. Note that the optimal architecture of the CNN was configured after a applying grid search over the hyper-parameters of the network, as summarized by the results in Table 3.1. The final CNN model, which represents the image-based component in our proposed multi-level IoT malware classification approach (Figure 3.1), consists of the following layers:

- **Input layer**. The input of the network is a $128 * 128 * 1$ image array of pixel values in the range $[0 - 255]$.
- **Convolutional layer**. The main building block of a CNN is the convolutional layer. It is responsible for applying various convolution filters (64) over the pixels to produce a feature map. Each convolution filter has specific height and width, in our case, 3 *x* 3, and by design

it covers the entire depth of its input. The final output of the convolution layer is a distinct feature map, put together by stacking all feature maps from multiple convolutions on the input. The activation function adopted is the *relu* function [105].

- **Pooling layer**. After a convolution operation, *pooling* is performed to reduce the dimensionality. This enables us to reduce the number of parameters, which both shortens the training time and fights overfitting. We apply *max pooling* which slides a window of size $2*2$ over its input, and simply takes the max value in the window. After the pooling layer, we perform a *dropout* of 2.5% to prevent overfitting. which makes the network perform better.

- **Fully-Connected layer**. After the convolution and pooling layers, we add fully connected layers to wrap up the CNN architecture. The output of both convolution and pooling layers are 3D volumes. Since a fully connected layer expects a 1D vector of numbers, we *flatten* the output of the final pooling layer to a vector, which becomes the input to the fully connected layer. Our first fully connected layer consists of 128 units, followed by a dropout of 0.5 and a second fully connected layer with 100 units. Our last fully connected layer combines the features learned by the previous layers and applies the *softmax* function to output the normalized probability distribution over malware families.

3.6.2 Evaluating the String-based Component

We compare the performance of the CNN and LSTM deep learning models to classify IoT malware based on the strings extracted from the malware. As presented in Table 3.1, the the 10-fold cross validation and evaluation results of the two algorithms demonstrate significant accuracy and F1-scores for both implementations (about 98%). Nevertheless, it can be observed that the CNN model implementation produced a relatively higher accuracy (98.86%) and macro F1-score (98.51%), thus, chosen as our candidate model for implementing the strings-based component within proposed IoT malware classification model (Figure 3.1).

The optimal architecture of the CNN string-based component, which was configured after a grid search over the hyper-parameters of the network (Table 3.1), consists of the following layers:

- **Embedding Layer.** This layer is defined as the first hidden layer of the network. It requires

that the input data be integer encoded, so that each word is represented by a unique integer. It takes as arguments the input dimension, i.e., the size of the vocabulary set to 50,000 words in our case, the output dimension (i.e., the size of the vector space in which words will be embedded (100)), and the input length (i.e., input sequences that have 400 words each).

- **Convolutional Layer.** The CNN's convolutional layer "scans" text which is organized into a matrix, with each row representing a word embedding, like it would an image, breaks it down into features, and judges whether each feature matches the relevant label or not. The chosen kernel size is 4, and the number of convolutional filters applied is 32. The activation function adopted is the *relu* function [105].

- **Pooling Layer.** A pooling of size 2 is applied to the input. The pooling stage reduces the dimensionality of the word features and retains only a simple probability score that reflects how likely they are to match a label.

- **Fully-Connected Layer.** At the final stage, these scores, flattened, are the inputs to a fully connected neural layer. The "fully connected" part of the CNN network goes through its own back-propagation process, to determine the most accurate weights. Each neuron receives weights that prioritize the most appropriate label. The activation function is softmax for multi-class classification.

3.6.3 Effectiveness of the Proposed Multi-Level DL Model

To answer our RQ1, we evaluate the effectiveness of the proposed multi-level deep learning model against the implemented image- and strings-based components, which are trained on each data modality independently. To do this, we leverage the implemented models for the image- and strings-based components to deploy our multi-level classifier and evaluate its effectiveness. We pre-train each component separately while utilizing their top learned features ($n = 100$ each) as an input to the feature fusion and classification component, as illustrated in Figure 3.1. The outcomes of the feature fusion and classification steps demonstrate the effectiveness of our multi-level IoT malware classification model with significantly high accuracy and F1-score that exceed 99.5%. Additionally, it is clearly observed that the multi-level model outperforms the DL implementation of the image-

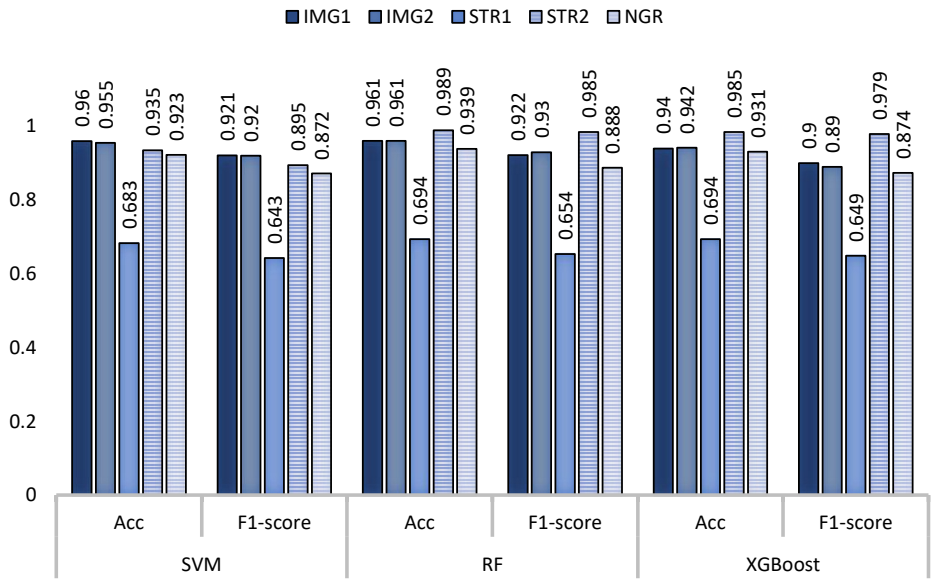


Figure 3.4: Evaluation results for the selected feature-engineering approaches.

and string-based components (Table 3.1). Moreover, looking at the results of Table 3.2, which summarizes previous state-of-the-art classification approaches, showcases the effectiveness of our approach compared to other related work. A complete overview of these related work is presented in Section 2.3. Thus, answering our first research question by demonstrating that the multi-level DL model that learns and combines characteristics of malware from various sources can yield better classification outcomes as compared to DL classifiers that rely on a single modality of data.

3.6.4 Comparison with Feature Engineering Approaches

To answer our second research question (RQ2), we perform image- and string-based classification using a set of diverse features that were extracted from the malware grayscale images and strings. While there are many possible combination of features and classification models, for the sake of comparison to our deep learning approaches, we reviewed the state-of-the-art malware classification approaches from the literature [55, 56, 59, 106] and identified combinations of features/-models that were more relevant to our experiments.

Image-based features. A malware binary is first converted to an image representation, as explained in Section 3.4.1, on which texture based features can be used as visual signatures for each malware family. We extract two sets of features from the grayscale image representation of

Table 3.2: Summary of the previous state-of-the-art classification approaches (NA stands for Not Available).

	Approach	Feature Type	Classifier	Env.	Dataset (#Samples)	Accuracy
Grayscale Image	Nataraj et al. [54]	GIST features	k-NN	Win.	Anubis (9,458)	0.9808
	Ahmadi et al. [55]	Local Binary Pattern features	XGBoost	Win.	BIG (10,868)	0.9724
	Gibert et al. [57]	128 x 128 Grayscale Image	CNN	Win.	BIG (10,868)	0.9750
	Su et al. [58]	64 x 64 Grayscale Image	CNN	IoT	IoTPOT (500)	0.9400
Strings	Tian et al. [59]	Printable String Information	Random Forest	Win.	Zoo (1,367)	0.9700
	Nguyen et al. [61]	PSI Graphs	CNN	IoT	*NA* (4,002)	0.9240
	Alhanahnah et al. [60]	Statistical & String features,	Multi-stage Clustering	IoT	IoTPOT (5,150)	0.9550
Integrated Features	Islam et al. [64]	FLF, PSI, API features	SVM, RF, DT, IB1	Win.	*NA* (2,939)	0.9705
	Ahmadi et al. [55]	IMG, STR, 1G, ENT, API, etc.	XGboost	Win.	BIG (10,868)	0.9977
	Mays et al. [63]	Image & Opcode N-Grams	EL (CNN, NN)*	Win.	BIG (10,868)	0.9724
	Gibert et al. [48]	APIs, Bytes & Opcode sequence	Multi-level Deep NN	Win.	BIG (10,868)	0.9975
	This Work	**Grayscale IMG, Malware Strings**	**Multi-level Model**	**IoT**	**IoTPOT (30,000)**	**0.9978**

*Ensemble Learning

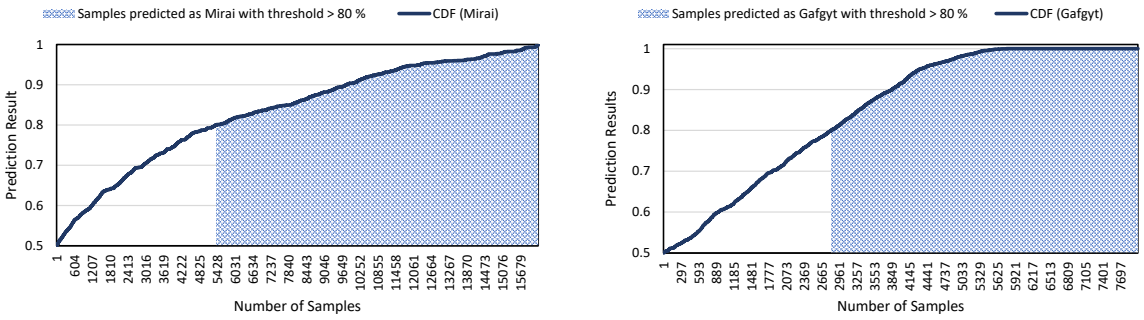
malware:

- *Haralick* features (**IMG1**), which compute a global representation of texture based on the Gray Level Co-occurrence Matrix, (GLCM), a matrix that counts the co-occurrence of neighboring gray levels in the image [106].
- *Local Binary Pattern* features (**IMG2**) instead, compute a local representation of texture which is constructed by comparing each pixel with its surrounding neighborhood of pixels.

String-based features. The combination of extracted printable strings from malware samples forms a kind of "digital fingerprint" for each malware family. We leverage the following feature sets for our comparison approach:

- Histograms related to the frequency distribution of length of strings (**STR1**) among different malware samples.
- *Bag-of-words* (**STR2**), by generating a global list of all of the strings that occur that are more than three bytes and their frequency.
- *N-grams* (**NGR**), where occurrences of n pairs (n=*2*) of consecutive strings are counted.

Implemented Classifiers. To perform the classification using the aforementioned features, we use Support Vector Machine (SVM), Random Forest (RF) and XGBoost, which have been proven to be consistently effective for implementing image- and string-based classifiers with feature-engineering [55, 56, 59]. The performance results of our adopted feature-engineering approaches after a 10-fold cross validation are presented in Figure 3.4. Overall, STR2 features resulted in the highest classification outcomes across the deployed models, with an accuracy of about 98.9% and 98.5% when using the RF and XGBoost classifiers, respectively. Despite such promising results, our implemented multi-level DL architecture outperforms all the tested implementations in terms of the overall classification accuracy and F1-score (Table 3.1). More importantly, using feature-engineering comes with practical limitations, which hamper the overall usability and performance of such approaches as compared to the end-to-end DL methods. For instance, the size of the *Local Binary Pattern* features, which increase exponentially with the number of neighbours, and the high dimensionality of the GLCM matrix used to extract *Haralick* features, lead to an increase of computational



(a) (b)

Figure 3.5: Cumulative distribution function of the samples predicted as (a) `Mirai` and (b) `Gafgyt`.

complexity in terms of time and space. Additionally, a drawback of *bag-of-words* and *n-grams* feature-engineering approaches is that they lead to a high dimensional feature vector due to the large size of strings vocabulary. Besides being tedious and time-consuming, manual feature engineering is problem-specific, error-prone, and limited by human expertise and capabilities. Thus, we answer our second research question by demonstrating that our multi-level DL approach is in fact more effective, scalable, usable, and practical, especially when used with a large amount of raw data.

3.6.5 Label Prediction for Unknown/Unseen Malware

We leverage the proposed multi-level IoT malware classification model in an attempt to identify the family labels for 24,271 unlabeled (unknown/unseen) malware samples in our dataset. To do this, we trained our multi-level classifier with 10,234 IoT malware samples from four predominant malware families (`Mirai`, `Gafgyt`, `Tsunami`, and `dofloo`). Note that despite the lack of ground truth for the unknown/unseen IoT malware, our supervised learning approach will generate multi-label classification outcomes which associate the analyzed samples to non-mutually exclusive malware family labels with a degree of confidence. Accordingly, we assume that a sample with high predicted class value is likely to belong to that specific known IoT malware family. On the other hand, a low class prediction value means that the analyzed sample is less likely to belong to that specific malware family.

The cumulative distribution functions (CDF) for the samples that are predicted as `Mirai` and

Gafgyt are illustrated in Figures 3.5a and 3.5b. It is worth noting that the majority of the samples within these two groups are in fact labeled with a relatively high prediction accuracy. Nevertheless, to avoid false positives and achieve a more reasonable/accurate labeling outcome, we consider a sample to be correctly classified as either of these two classes if and only if it was predicted with a threshold greater than 80%. Accordingly, for the first group (Figure 3.5a), we classify about 67% of the samples (10,786 out of 16,214) as Mirai while about 55% of the the malware samples within the second group (Figure 3.5b) (4,332 out of 7,923) were classified as Gafgyt.

Moreover, our analysis of the Tsunami-labeled malware samples shows that about 80% of them (108 out 134) were predicted with high confidence (>80%). Interestingly, our classification outcomes did not associate any of the unknown/unseen malware samples with the Dofloo family. This might be due to the fact that our unlabeled dataset was mainly obtained from IoTPOT, which is a specialized IoT honeypot that is designed to interact with Telnet requests (Section 3.7), hence, it did not capture Dofloo attacks since they target other ports/services (e.g., TCP port 2375) [107]. Additionally, it is possible that the Dofloo malware is not actively circulating in the wild due to the specificity of its implementation and targeted vulnerabilities, which have been addressed, respectively. Confirming this however, is beyond the scope of current work and will be considered for future work.

Results Validation. It is important to realize that obtaining the ground truth in terms of malware family labels for the unknown/unseen malware samples is a challenging task since these samples have not been detected or known by major antivirus vendors. To overcome these challenges, we perform a targeted text similarity analysis on the extracted strings from a random sample of IoT malware that were classified by our multi-level model with high confidence (i.e., $>$ 80%). We use regular expressions to obtain special textual indicators associated with known malware families while correlating the strings extracted from the unknown malware samples to those known ones through the strings-based similarity analysis. Our assumption is that malware samples from the same family are likely to share string-based indicators that reflect their underlying implementations, functionalities, targeted services, adversarial IP addresses, and command instructions.

Note that the Mirai family consists of the largest number of malware samples in our dataset. Therefore, we validate the outcomes of our classifier by manually inspecting 10,786 Mirai-labeled

malware samples and performing text-based similarity analysis on them to associate them with known Mirai samples. To do this, we leveraged known Mirai samples to identify adversarial IP addresses that might be associated with possible C&C servers or targeted victims. Additionally, we identify other possible Mirai indicators such as pre-configured default usernames and passwords that are used during brute-force attacks, commands sequences for communication with the C&C server, Internet scanning/probing instructions and commands, DDoS attack commands, and downloaded malicious payloads/scripts, to name a few.

Our analysis of the Mirai-labeled samples resulted in identifying 3,378 unique IP addresses, among which, about 67% were matching IP addresses extracted from known Mirai samples. While common adversarial IP addresses across different malware samples can associate those samples to the same adversary (e.g., bot master), it might not necessarily mean that they belong to the same malware family. Therefore, we look for additional Mirai-specific string indicators within the analyzed samples [2, 108]. Interestingly, about 95% of the Mirai-labeled samples contained one or more instances of Mirai-specific strings such as "`POST /cdn-cgi/`", "`/dev/misc/watchdog/`", "`.mdebug.abi32`", "`LCOGQGPTGP`", and "`CFOKLKQVPCVMP`", to name a few. We also found similar commands that are used by Mirai to check for "`wget`" or "`tftp`" installations before using them to downloading and executing further scripts and attack payload from designated servers. Since Mirai is designed to launch DDoS attacks, we also found attack methods such as "`attack_udp_dns`", "`attack_udp_vse`", "`attack_tcp_stomp`", "`attack_tcp_syn`", and "`attack_tcp_ack`", in the analyzed malware strings.

Despite that our validation approach using strings similarities was tested with one IoT malware family (Mirai), our findings shed light on common string/textual indicators that can be used to attribute unknown malware samples to those from known families.These findings can be used to answer our final research question (RQ3), while demonstrating high levels of confidence in the classification outcomes with respect to labeling unseen/unknown IoT malware samples.

Undecisive Labels. In addition to the unknown malware samples that were classified with high confidence, a total of 4,773 samples were labeled with a relatively low prediction values ($\leq$70%), meaning that their labeling is positively indecisive. Since the model can only predict four family labels, an indecisive and low prediction result may indicate that these samples might belong to other

known malware families, or are possibly new variants/families, which were not incorporated in training the classifier. Additionally, such results could be due to deployed binary obfuscation and/or possible corrupt files, which hide or scramble the binary contents and lead to possible misclassification outcomes. To overcome these challenges, one can investigate various malware de-obfuscation techniques, while extracting meaningful information that can be used for further similarity analysis in correlation to samples from known malware families.

3.7 Discussion

In this paper, we use next-generation techniques, combining multimodal end-to-end learning and malware features from multiple sources (e.g., static malware analysis), to classify IoT malware executables and attribute them to known families. Yet, while some previous works that rely on transfer learning [65, 66] and a variety of complex features such as control flow graphs [61, 67, 68], API function calls [48, 55], have been shown to be effective in classifying malware samples, these solutions mainly rely on the ability of domain experts to extract meaningful features. Additionally, these approaches often rely on deep neural networks with hundreds of layers, such as ResNet50 and ImageNet, which are computationally heavy and thus, difficult to handle by resource-constrained IoT devices. While this is proven to be an expensive task, we leverage multiple deep learning models to implement a lightweight multimodal learning approach that provides a better or comparable classification accuracy, while enabling scalable and timely classification without any pre-processing or feature engineering.

In addition, we demonstrate the effectiveness of our proposed multi-level model in preventing the co-adaptation of the sub-networks to a specific feature type, therefore making the network less sensitive to the loss of one or more channels of information. Our intuition is that in a real world scenario, it is not possible to rely on one malware characteristic, as some features might be difficult to obtain (e.g., due to obfuscation), or insufficient to differentiate between two malware variants. This is indeed the case with IoT malware, where a significant amount of malware samples in the wild are found to share similar implementations and functionalities due to wide code-reuse, as observed in the case of `Mirai` and the consequent IoT malware that were based on `Mirai`'s source code [44].

Another advantage of our proposed classification framework is in its modular architecture, which can be complemented by new features from other data modalities to make it adaptable to new contexts (e.g., ransomware). Moreover, such adaptability to various contexts can positively support the generalizability of our approach and the obtained findings. To prove this, further experimentation using various types of malware is required, which can be considered for future work. Moreover, we show the generalizability of our approach to unknown samples and its ability to predict the labels of unseen malware (see Section 3.6.5).

A main outcome of this work is to draw attention to the limitations of existing malware detection and labeling outcomes provided by major antivirus vendors. Specifically, we identify a considerable number of IoT malware samples, which have not been detected or labeled by the antivirus vendors. Consequently, we demonstrate the effectiveness of our proposed IoT malware classification approach to address this limitation, while predicting reliable family labels for such unknown/unseen samples. In addition, our findings show that the majority of those new/unlabelled malware samples were in fact associated with predominant malware families such as `Mirai` and `Gafgyt`. These findings highlight a common practice among IoT malware authors, who often rely on reusing/tweaking existing malware implementations to create new malware instances that can be used to target new vulnerabilities in a timely manner.

In line with that, our extended analysis of the new/unknown IoT malware samples unveiled `Covid-19` themed `Mirai` variants, which included different string indicators such as "`/bin/busybox CORONA`" command, or "`Total lockdown is the solution`". More importantly, considering that this work was conducted during the `Covid-19` pandemic, the identification of such covid-related IoT malware samples shed light on the rapid evolution of IoT malware, which are designed to abuse global events to their advantage. Therefore, this work contributes to the cybersecurity research by providing means for timely classification of new/unknown IoT malware while attributing them to known families (when applicable).

Finally, it is important to realize that addressing threats associated with the emerging IoT malware is essential for the security of the Internet ecosystem. Moreover, the deployment of next-generation 5G networks and technologies will enable large-scale deployment of IoT devices to support everyday activities of consumers and service providers. Therefore, given the insecurity of

the IoT paradigm, the projected increase in the deployed IoT devices will without a doubt amplify the threats associated with IoT malware and IoT-driven cyber attacks. To mitigate such threats in the context of future networks, we envision the integration of our next-generation malware analysis and classification approaches within the intermediate cloud-based monitoring and management systems, to support accurate and real-time malware detection, classification, and attack mitigation, and thus, contributing towards the overall security of the 5G networks.

3.7.1 Limitations

A main limitation of this work is to collect a diverse and representatives dataset of a IoT malware samples, which is a challenging task. Indeed, relying on a single source for data might hinder the diversity and generalizability of the obtained IoT malware dataset. Nevertheless, while we leveraged IoTPOT [8] as our main source of data collection, it is worth noting that IoTPOT is considered as one of the most reliable sources of IoT malware data, while providing access to a large number of diverse and recent IoT malware samples. It has also been shown to be more effective than other honeypots (e.g. Honeyd [109]) at capturing various IoT-tailored attacks. In addition, the analysis of Internet-scale scanning activities generated by compromised IoT devices [40, 41, 53] confirms the prevalence of `Mirai`-like malware attacks/samples in the wild, which are the most dominant variants in the IoTPOT dataset. Additionally, to address the diversity of the data and collect a more representative dataset, we extended our data collection to obtain further IoT malware samples from well-known threat repositories such as VirusTotal and VirusShare. Accordingly, our final dataset contained a representative dataset with more than 70,000 IoT malware samples belonging to four predominant families (`Mirai`, `Gafgyt`, `Tsunami`, and `Dofloo`).

Another limitation of this work is that we did not consider obfuscated IoT malware samples in the implementation of our proposed malware classification approach. It is important to note that malware authors are inclined to intentionally conceal their identity and therefore use obfuscation techniques to hide data in a malicious executable binary [98]. Therefore, such factors must be considered while building a robust malware classification model, which can also deal with obfuscated samples. Nonetheless, our analysis of the IoT malware samples showed that the majority of them did not employ sophisticated obfuscation, thus, were de-obfuscated using off-the-shelf tools (e.g.,

UPX). Therefore, our proposed model is still capable of performing effective classification with respect to the majority of IoT malware samples, while attributing them to known families with high degree of confidence.

3.7.2 Future Work

The continuous evolution of malware variants/families might negatively impact the prediction outcomes of deployed ML classification models over time (concept drift). In the context of IoT, the release of the `Mirai` source code to the public enabled adversaries to reuse the code while creating new `Mirai`-like malware variants by incorporating new exploits to the existing code [44]. Hence, creating a corpus of malware samples with new versions to share similarities with older versions, respectively. Nevertheless, these similarities are assumed to degrade slowly, while causing the malware population to drift over time. Accordingly, we believe that the prediction quality of malware detectors and classifiers will eventually decay in the future as malware evolution might result in completely new variants [92]. As a result, we study the aforementioned issue of concept drift in our next thesis contribution as presented in Section 4.

Another interesting future research direction is to investigate the problems associated with adversarial ML, where an attacker is assumed to manipulate data and craft adversarial examples using different techniques (e.g., manipulation of static feature) to deceive the detection/classification model [98,110]. While our proposed framework achieves accurate classification of IoT malware, we believe that future research is required to examine the robustness of our multi-level approach against adversarial ML techniques and detection evasion methods. For instance, improving interpretability can increase our multimodel's robustness to adversarial attacks, by revealing and understanding which features give more weights to the model's performance, and therefore by crafting AEs generated based on these features-level manipulation.

3.8 Summary and Concluding Remarks

In this work, we utilized DL architectures to implement and evaluate a novel approach for classifying IoT malware by combining multi-dimensional features extracted from strings- and image-based representations of the executable binaries. Moreover, we addressed the main challenges associated with feature selection/engineering by implementing an end-to-end DL approach that can automatically extract and meaningfully combine features from different representation of the analyzed IoT malware binaries. Our experimental results using 10,234 IoT malware samples from four prominent families demonstrated the effectiveness of our classification approach, with a significantly improved accuracy (99.78%), as compared to conventional single-level approaches. In addition, we demonstrated the capability of the implemented model towards classifying new IoT malware binaries, which were mainly attributed to few known families (e.g., `Mirai` and `Gafgyt`). Finally, considering the projected increase in the number of deployed IoT devices, and the pivotal role of these insecure devices in the operation and infrastructure of next-generation 5G networks, this work provides a major step towards the development of practical data-driven tools/techniques for effective IoT threat detection and mitigation, thus, contributing to the security of the IoT ecosystem.

Chapter 4

EVOLIoT: A Self-Supervised Contrastive Learning Framework for Detecting and Characterizing Evolving IoT Malware Variants

4.1 Overview

The recent growth in the number of IoT devices has motivated the rise of IoT-tailored malware that enable cyber-attacks as part of coordinated and monetized large-scale botnets [7]. The public release of the source code of some of the main IoT malware families such as the `Mirai` [2], has forever shaped the IoT threat landscape. Specifically, the effectiveness of `Mirai`, and the ability to add new exploits to the codebase has paved the way for more advanced and sophisticated `Mirai`-like malware variants such as *Hajime* [111] and *Satori* [112], to name a few. The increasing number of detected IoT malware and the threat associated with the IoT-driven cyberattacks has pushed the security community to focus their effort on deploying specialized honeypots [34, 36, 38] to detect

This work has been submitted to ACM ASIACCS 2022.

IoT malware/botnet and investigate their inner operations [2, 6, 7]. This is essential for developing rigorous mitigation approaches against the spread of IoT malware strains by building effective learning-based malware detectors and classifiers [10, 58, 113].

Despite such effort, the complex relationship and similarities in terms of code reuse among malware variants bring several challenges for malware analysis including labeling, provenance, triage, lineage analysis, as well as family and authorship attribution. In fact, it is unclear what makes each group of malware distinct, what links together popular families, or how the same malware family evolves over time. This is also reflected by the labels assigned by anti-virus (AV) vendors, which are often inconsistent and coarse-grained, and often unable to capture the code reuse between IoT malware and their evolutionary characteristics.

More importantly, the rapid evolution of the IoT threat landscape along with the characteristics of newly detected IoT malware variants in terms of their massive code reuse and underlying relationship [6, 44], is most likely to cause the performance of classifiers to degrade with time, as old malware campaigns are typically retired/updated while new ones are developed [85]. This change in the data distribution of a machine learning model is called concept drift [85], which makes it challenging to generalize existing learning models that were trained with older data to new, previously-unseen samples.

In order to build sustainable models for IoT malware classification, it is important to identify when the model shows signs of aging, by which it fails to effectively recognize new variants and adapt to potential changes in the data, especially when accounting for in-class evolution of IoT malware families. Thus, to build an effective and robust classifier, we must aim at detecting drifting IoT variants within the same malware family (i.e., in-class evolution) and consequently, interpret the meaning behind the drift to identify which mutations distinguish one variant from another. Note that previous research relied on the prediction decisions of a learning model as a by-product of the classification process [10, 88, 89]. However, it is likely that a drifting sample that does not belong to any class will be assigned with high confidence to the wrong class (i.e., closed-world assumption), which has been previously validated.

To mitigate this confidence bias, calibrated probability predictors (e.g., Venn-Abers Predictors [91]) as well as statistical non-conformity measures [90] have been proposed. Although useful,

these approaches cannot draw concrete conclusion on drifting/evolving samples and lose their effectiveness on high dimensional data. A more recent work by Yang et al. [95], used an auto-encoder coupled with a contrastive loss to compress the data and learn an effective distance between samples of different classes. While their resulting distance function can efficiently detect and rank drifting samples from distinct classes, their approach is not tailored to in-class drifting samples, which is more relevant in the context of IoT. Alternatively, a plethora of works studied the evolution of malware binaries by computing the similarity between functions extracted from the decompiled binary code (e.g., instructions), basic blocks, control flow graphs (CFGs) [77, 80–82, 84], and execution traces [69, 114]. While they provide invaluable insights, none of them is applied on Linux-based IoT malware. Cozzi et al. [44] took this opportunity to identify code similarities between IoT malware families using function-level binary diffing using off-the-shelf tools that are not tailored for IoT. Additionally, their approach required a substantial amount of manual adjustments, which hampers its scalability and feasibility for real-time threat detection and analysis.

In this work, we aim at filling this gap by detecting evolving IoT variants and understanding their evolutionary trajectories, in a systematic and scalable way tailored to the peculiarities of IoT malware. We propose EVOLIoT, a self-supervised contrastive learning approach based on pre-trained language models such as BERT [11], which effectively learns and compares semantically meaningful representations of binary code, without the need for expensive target labels. This presents an immense advantage to the problem of scarcity of labeled data (e.g., emerging IoT malware) in security applications. In fact, the proposed contrastive objective views the evolution of IoT malware binaries as a natural language augmentation strategy, and show how it can be used as a representation learning objective which maximizes the mutual information between malware sequences and their conserved malicious function. As such, EVOLIoT identifies evolved samples by constructing a "positive" pair through feeding the same sample to the encoder twice to get two embeddings that only differ in hidden dropout masks found in the Transformers [115]. As such, the model learns to predict positive pairs among other embeddings (i.e., negative pairs). In other words, EVOLIoT learns to encourage "disagreement" across evolutionary views by contrasting the rest of the embeddings, and thus, learn to discriminate between samples coming from the same class, by maximizing the distance to drifting embeddings instead of minimizing it.

4.2 Contributions

To this end, we make the following main contributions:

- We leverage the power of contrastive learning to address concept drift and the limitations of inter-family IoT malware classification due to the evolution of IoT malware. We present EVOLIoT, a robust and effective contrastive method that learns and compares semantically meaningful representations of IoT malware binaries without the need for expensive target labels.

- We are among the first to address the limitations of inter-family classification and attribution of IoT malware binaries using contrastive learning. We propose to view the evolution of IoT malware binaries as a desirable choice of augmentation to construct "views" for contrastive learning in a security application, from both a theoretical and technical point of view. We illustrate that our contrastive learning objective, which is based on evolutionary augmentation, directly encourages representational invariance to shared features across positive views while at the same time, encouraging disagreement across views by dealing with same-class augmentations as negative to each other.

- Motivated by the number of IoT samples, their diverse target architectures and their general lack of obfuscation, we adopt a cross-architecture code-based analysis that can capture a binary's malicious intent and evolutionary essence. Our framework leverages the cutting-edge BERT architecture [11] to deeply infer the underlying code semantics regardless of the binary's instruction set architecture (ISA). We also consider a well-balanced instruction normalization strategy that strikes a balance between too-coarse grained and too-fined grained normalization, to conserve as much contextual information as possible for cross-architecture syntactic variations while maintaining efficient computation.

- We evaluate our approach using a large corpus of IoT malware binaries that were detected over a course of 3 years. We leverage an interpretable strings-based analysis to detect and validate more than 50 variants belonging to the top 3 IoT malware families: `Mirai`, `Gafgyt`, `Tsunami`. Our analysis highlights the constant evolution of variations among each family

from different perspectives such as the added target exploits and 0-day vulnerabilities, TOR-enabled botnet communication, and botnet behaviors (e.g., detection evasion), to name a few.

- We extensively evaluate our proposed method on three applications, by leveraging EVOLIoT as a semantic search engine for finding semantically similar variant queries, and testing the effect of our balanced normalization process and our cross-architecture embeddings in reducing out-of-vocabulary instructions and preserving semantics, respectively.

- We make our ground truth dataset with identified fine-grained labels, as well as the full list of identified evolutionary trajectories (e.g., exploits, variants) available[1] to researchers to support future work.

4.3 Background and Problem Scope

In what follows, we elaborate on the problem of concept drift in the context of security applications (e.g., malware evolution) and highlight the power of attentive and self-supervised methods as effective solutions.

4.3.1 Concept Drift (In-Class Evolution)

Concept drift has been used to describe the problem of the changing relation between the input data and the target variable over time [85].

In cybersecurity, these changes apply to the malware development and data generation process where attackers are constantly modifying their attack vectors, trying to bypass defenders' solutions [116]. Moreover, the evolution of malware is another problem related to this challenge, where the process of defining and improving variants results in new types of attacks. Concept drift in a multi-class classification setting usually involves drifting samples from previously unseen families (new class), or drifting samples from an existing class but with changing behavioral patterns (in-class evolution).

In this paper, we focus on the problem of *in-class evolution*, which is understudied in literature and more relevant in the context of IoT malware. Specifically, we examined the in-class evolution of

[1] https://github.com/IoTMalw/EVOLIoT

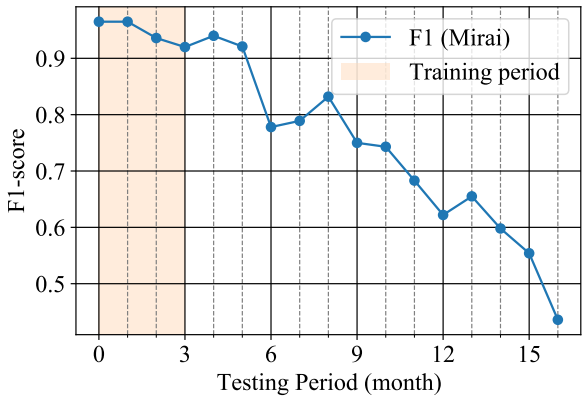


Figure 4.1: The impact of drifting samples on the classification accuracy for samples of the `Mirai` family (trained/tested MLP classifier with a 3 months sliding window).

samples from the `Mirai` family over one year by training an MLP classifier on samples detected in the first 3 months while testing with previously-unseen samples detected in the following 3 months intervals, respectively. As illustrated in Figure 4.1, we capture the impact of the drifting samples through examining the degrading accuracy of the MLP classifier over time (detailed in Section 4.5.2).

In fact, previously proposed ML-based solutions for detecting concept drift [85] are not necessarily suitable for security applications as they mainly rely on the collection of new sets of well-prepared and labelled data to statistically assess model behaviors [87,91,117]. However, in the context of cybersecurity, new attacks/data are usually unknown thus, it is almost impractical to assume that the incoming data is sufficiently labeled for (re)training classification models. Moreover, data labelling is usually time-consuming, expensive, and requires expertise. Instead, we focus on a more practical scenario that leverages contrastive learning, as explained in the following sub-section.

4.3.2 Contrastive Learning (CL)

Contrastive learning is a type of Self-supervised Learning (SSL), which allows the model to learn sentence-level semantics by comparison. In general, SSL has emerged as a powerful method for learning effective representations without the need for expensive target labels [11]. This presents an immense advantage to the problem of scarcity of labeled, clean data (e.g., malware) in cyber security applications. Moreover, it helps a model gain more generalization ability by learning from

large amounts of unlabeled data, in contrast to a supervised model which learns only from what is available in the training data.

To this end, we rely on contrastive learning, which works by pulling semantically close neighbors together and pushing apart non-neighbors. Recent progress on self-supervised contrastive methods has proven its effectiveness in learning good data representations for instance discrimination and semantic similarity tasks in various domains such as computer vision [4, 118], audio processing [119], and computational biology [120, 121]. Hence, we explore the idea of contrastive learning to learn semantically-aware binary code representations of IoT malware variants to detect their mutations and evolution (Section 4.4). Moreover, to effectively learn sentence embeddings from unlabeled data, we incorporate pre-trained Attention and Transformer language models such as BERT [11], as described in the following sub-section.

4.3.3 Attentive Transformer Language Model

BERT's model architecture is a multi-layer bi-directional Transformer encoder based on the original implementation described in [122], which provides rich vector representations of a natural language by capturing the contextual meanings of words and sentences using a multi-head self attention mechanism [123]. It consists of two main processes: (a) a *pre-training* phase, based on masked language model (MLM) and next sentence prediction (NSP) strategies to build a generic model that considers context and orders of words and sentences in a large data corpus; (b) a *fine-tuning* process that applies the pre-trained model to a specific downstream task. Sentence-BERT (SBERT) [124], proposed as a modification of the pre-trained BERT network, uses siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be efficiently compared using cosine-similarity. In this work, we demonstrate that a contrastive objective, coupled with pre-trained language models such as BERT [11], can be extremely effective in learning sentence embeddings from unlabeled data.

4.3.4 Problem Scope and Insights

The widespread use of packing and obfuscation made static analysis generally inadequate in the malware domain [125]. However, to our advantage, the current number of samples and the general

lack of sophisticated obfuscation in IoT binaries enable code-based analysis [6, 44]. Conveniently, modelling the binary code of an executable can provide a precise reflection of its malicious intent and evolutionary essence, especially in terms of code reuse and added functionalities. Hence, identifying evolving malware strains can be equivalent to a binary similarity comparison problem, where two samples have syntactically or semantically (dis)similar feature vectors. Specifically, the equivalent semantics of a binary code can be defined as a sequence of instructions that carries out an identical task from a logical function in the original source. Yet, the realm of IoT presents a few challenges:

(1) Multiple efforts are in place to ensure that IoT malware samples can run on devices with diverse hardware architectures and system configurations. Meaning, two binaries compiled from the same source code for different architectures (e.g., ARM, MIPS) can present syntactically different instructions.

(2) In contrast to previous work [77, 81, 84, 126, 127], the code equivalence problem cannot be resolved at the *function*-level due to the tangled relationships of similarities and code reuse between IoT malware binaries. In fact, "stolen" code components from a function or a set of functions can be inserted into other code, that is, borrowed code is not necessarily a function.

(3) Identifying the start of functions within binary code is a common problem in the context of static malware analysis. In fact, the performance of well-known analysis tools such as IDA Pro has been shown to deteriorate significantly (drops to 60%) when identifying the start of functions in stripped binaries [45], whereas it works consistently for correctly identifying instructions and basic blocks.

In fact, the key to identifying differing IoT malware variants is to explain their evolutionary trajectories regardless of their target instruction set architectures (ISAs). Therefore, considering the above-mentioned challenges, determining the similarity between two malware variants requires a precise and efficient cross-architecture embedding model, which can capture the semantics and dependencies of instructions.

While previous solutions failed to procure such results, we proceed by regarding instructions as words and basic blocks as sentences. Particularly, we present a new method to train such sentence

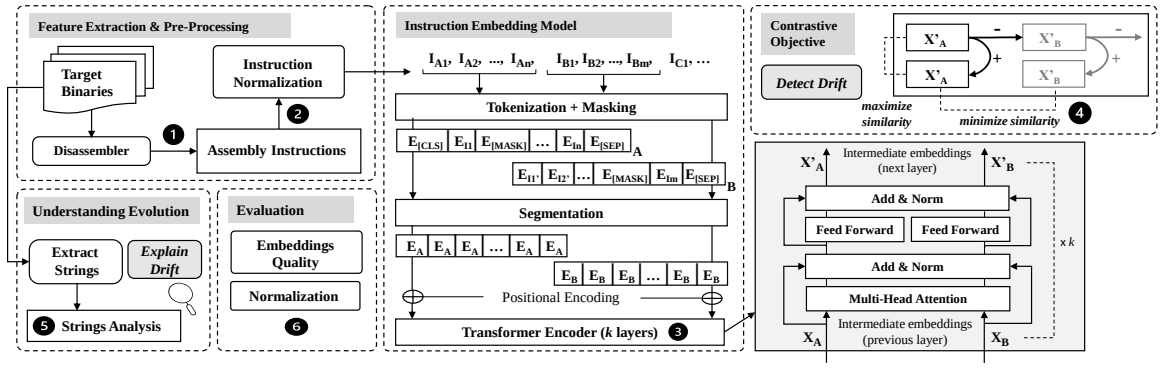


Figure 4.2: An overview of the proposed EVOLIoT framework/approach and its various stages.

embeddings without relying on training data, and by leveraging Sentence-Transformer and contrastive learning. The idea is to consider the evolution of IoT binaries as an augmentation strategy, and hence encode the same instance twice to form a *positive* pair. The distance between these two embeddings will be minimized, while the distance to other embeddings of the other sentences in the same batch will be maximized (i.e., they serve as *negative* pairs). An overview of our proposed approach is detailed in Section 4.4.

4.4 Approach

In this work, we propose a new approach to address the problems of concept drift and inter-family IoT malware classification by leveraging contrastive learning. We have two main objectives: (i) Detecting in-class evolving/drifting IoT malware binaries, and (ii) interpreting the meaning behind the drift. To achieve our objectives, we follow a multi-stage methodology, as illustrated in Figure 4.2. In particular, we detect drifting IoT binaries by extracting a feature modality from the malware binaries (e.g., assembly code), normalizing all instructions, learning a vector to represent the semantic meaning of the assembly code of the executable binary, and then deriving an effective distance function to measure the dissimilarity of samples. Second, the goal of interpreting the drift is to identify the causes of the drift (e.g., added functionalities, behavior changes, etc.) and link the detection decision to semantically meaningful features.

In what follows, we provide further details about each stage of the proposed methodology (Figure 4.2).

4.4.1 Feature Extraction & Pre-Processing

We start our analysis by extracting and normalizing important features (e.g., assembly instructions) that carry enough information to effectively differentiate a drifting sample from another.

Binary Code Representation. Formally, the assembly code of an executable is a set of assembly functions. An assembly function f is in turn a set of basic blocks each containing a sequence of assembly instructions, denoted by $I_f : (i_1, i_2, i_3, ..., i_m)$, where m is the number of instructions in the function. These sequences of machine instructions are analogous to a natural language, which implies the possibility of utilizing effective techniques in an NLP domain such as BERT for a binary task. In fact, the authors of InnerEye [82] apply the idea of Neural Machine Translation (NMT) to a binary function similarity comparison task by regarding instructions as words and basic blocks as sentences.

We start by extracting a set of essential artifacts ❶ (e.g., feature modalities) from our malware samples (Table 2.2), using commercially popular static binary analysis tools. We leverage IDA Pro [128] to extract basic blocks of assembly instructions.

Assembly Instruction Normalization. Following that, we perform *Instruction Normalization* to map instructions to various tokens, which are leveraged by most deep learning methods to generate input sequences for their training phase. Hence, we propose a well-balanced instruction normalization ❷ that is neither too generic nor too specific, to avoid an *out-of-vocabulary* (OOV) problem, while capturing code semantics with tokens that hold rich information.

In fact, various approaches [80,82,84,126] have adopted mechanical conversions where the most common one is replacing every immediate operand with a single notation such as `immval`, without a thorough consideration of their contextual meanings. Such a coarse-grained normalization renders every call instruction identical, hence loses a considerable amount of contextual information. However, retaining immediate values [126] can raise an OOV problem due to a massive number of unseen instructions (tokens).

Conveniently, to maintain contextual and semantic information, good word embeddings must

rely on an individualized normalization strategy where binary code semantics are expressed as precisely as possible while using a reasonable number of tokens. For instance, it is important to differentiate between immediate values because an immediate can represent either a call invocation, a memory reference to jump to, a string reference, a statically-allocated variable, etc. Therefore, erasing such differences makes an embedding rarely distinguishable from another.

To this end, combining successful normalization strategies from the literature, we make sure to: (i) differentiate between a jump and a call destination, a string value, or a memory reference, (ii) consider different sizes in a 32-bit register [84], (iii) keep stack pointers or base pointers intact [126] while preserving pointer expressions to maintain memory access information. Finally, Opcodes are not normalized and are retained as they are.

4.4.2 Instruction Embedding Model

To this end, we leverage a pre-trained language model, namely BERT [11], to encode the normalized assembly instructions ❸ and then fine-tune all the parameters using the contrastive learning objective (§4.4.3). Note that BERT takes as input a token "sequence" which can either be single sentence or a pair of sentences packed together. We consider a sentence to be a block of assembly instructions. Such input representation allows BERT to handle a variety of down-stream tasks and thereby later serves our contrastive objective (§4.4.3).

We start from pre-trained checkpoints of BERT:

- We include a special token at the beginning of every sequence (`[CLS]`) to indicate the start of a sequence. In addition, we differentiate sentence pairs packed together into a single sequence by separating them with a special token (`[SEP]`). For instance, (`[CLS]` $W_1 W_2$ `[MASK]` ... W_n `[SEP]` $W_1 W_2$ ... `[MASK]` W_m), where W is a word in a sentence.

- We adopt BERT's original masked language model (MLM), which masks a percentage (e.g., 15 %) of the input tokens at random, and then predicts those masked tokens during pre-training. An advantage of masking is forcing the transformer to remember the context representation for every input token while hiding the words that it will be asked to predict at the final layer.

- BERT includes a next-sentence prediction (NSP) task which allows it to learn relationships between sentences by predicting if the next sentence in a pair is the true next or not. However, the semantics of a function should be considered *location-independent* from that of its adjacent functions. As such, attackers can copy/borrow a certain function from another source code and add it anywhere in their code. Therefore, we do not use the NSP task of BERT.

4.4.3 Contrastive Objective

In this section, we present the contrastive embedding framework ❹ behind EVOLIoT, which can produce superior "sentence" embeddings (i.e., assembly instructions feature vectors) from incoming unlabeled malware data, and learn how to compare them to identify differing variants within the same family. The idea of contrastive learning is to learn a good representation of unlabeled data by distinguishing similar samples from the others. It assumes a set $D = \{x_k^+\}$ including paired examples x_i and x_j which are semantically related, also referred to as *positive* pairs. Then, a neural network base encoder $f(.)$ extracts representation vectors from the data examples, denoted as $z_i = f(x_i)$ and $z_j = f(x_j)$. As described in §4.4.2, we adopt the pre-trained language model BERT as our encoder. Moreover, the contrastive objective is to identify x_j in the set of negative examples $\{x_k^-\}_{k \neq i}$ for a given x_i. Let $sim(z_i, z_j) = \frac{z_i^\intercal z_j}{||z_i||.||z_j||}$ be the cosine similarity between two feature vectors, the contrastive loss is defined as [4]:

$$\ell_{i,j} = -\log \frac{\exp(sim(z_i, z_j) \,/\, \tau)}{\sum_{z_k \in Z^-} \exp(sim(z_i, z_k) \,/\, \tau)}, \tag{4.1}$$

where $\tau \in \mathbb{R}$ denotes a temperature hyper-parameter ($\tau = 0.05$) to adjust the scaling of the similarity scores, and $Z^- := \{z_k\}_{k \neq i}$. This loss, which is computed across all positive pairs (i, j) and (j, i) in a mini-batch, brings the anchor and positive samples together while driving the anchor and negative samples apart.

Evolution as Choice of "Views". The critical question that follows is how to select "views" of the input, i.e., how to construct (x_i, x_j) positive pairs. In visual representations, Chen et al. [4] constructed such positive pairs $x_i = T_1(x)$ and $x_j = T_2(x)$ by taking two independent augmentations or "views" of a query image x from a pre-defined family of transformations T, where $T_1, T_2 \sim T$. Some

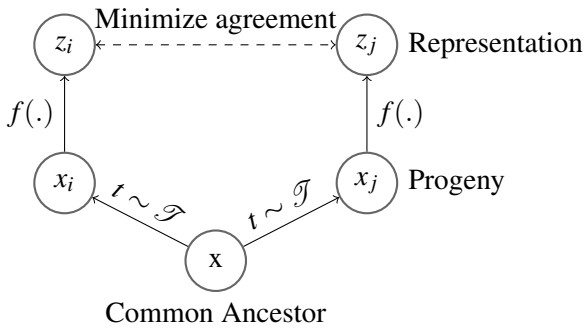


Figure 4.3: Re-casting SimCLR [4] as a phylogenetic tree where augmentations are the evolved malware variants.

frequently used image transformations are rotation, cropping or flipping. Recently, language and sentence representations have adopted augmentation techniques such as word deletion, substitution, and reordering [129, 130]. Moreover, in most cases, (x_i, x_j) are collected from supervised datasets. However, when it comes to malware binaries, data augmentation is inherently difficult because of their distinct nature, and incoming malware data is mostly unlabeled.

In this work, we consider the evolution of IoT binaries over the years as a theoretically and technically desirable augmentation strategy to construct "views" of the input. As described in Figure 4.3, IoT malware variants can be considered as "evolutionary augmented views" of a common ancestor x (e.g., first `Mirai` variant to appear), while $\mathcal{T}$ can denote possible evolutionary trajectories characterized by changing features and mutations (Figure 4.3). Much as x_i and x_j can be seen as two malware variants sampled from the same family at different times. For instance, x_i can be a variant of the `Mirai` family that have appeared in 2018, while x_j can be a variant of the same family that have appeared in 2020.

The key idea is that properties of the ancestral sequence will be preserved in both descendants (i.e., views). Therefore, by training a contrastive encoder to project them to nearby locations in the latent space, their proximity is ensured to correspond to similar malicious functions, even without explicit labels.

In general, contrastive learning encourages "agreement" between important features across evolutionary views. In contrast, our contrastive objective aims at identifying factors that contribute

instead to the "disagreement" between evolutionary views. In particular, while the existing contrastive schemes act by pulling all augmented samples toward the original sample, we suggest to additionally push the samples with shifting transformations away from the original. Namely, instead of considering evolved variants as positive to each other, we attempt to consider them as negative if they belong to the same malware family. The aim is to learn robust semantic representations that can capture small input variations between variants belonging to the same class.

Learning a good alignment for positive pairs. Considering the unsupervised nature of our dataset, an effective solution is to take a collection of input samples $\{x_i\}_{i=1}^{N}$ and use $x_i^+ = x_i$. The key idea is that the evolutionary trajectories or mutations within the same malware family are unknown, specially since ELF binaries do not contain a timestamp of when they were compiled as opposed to generic malware. Therefore, one might rely on public forum discussions or on the VirusTotal first submission time as ground truth. However, online discussions are time-consuming and difficult to track and might not contain accurate or reliable information. On the other hand, anti-virus engines' scan time of the binaries might not coincide with the time they actually appeared in the wild, as malware can go a long time before being detected.

Hence, one way to identify the evolved samples is to construct two different embeddings as *"positive pairs"* by feeding the same sample twice to the encoder and getting two embeddings that only differ in hidden *dropout* masks, found in Transformers [115]. We denote $z_i^h = f_\theta(x_i, h)$, where h is random mask for dropout. Thus, the distance between these two embeddings will be minimized, while the other embeddings in the same batch will serve as *"negative"* examples, and the model will predict the positive one among negatives. By contrasting the rest of the embeddings, the model will learn to discriminate between samples coming from the same class, by maximizing the distance to the shifting embeddings instead of minimizing it.

Predicting the input sentence itself with only *dropout* used as noise has been shown to greatly outperform training objectives such as predicting next sentences, discrete data augmentation (e.g., word deletion and replacement) and even matches supervised training objectives [115]. We verify the effectiveness of our contrastive objective for the detection of in-class evolution in our experimental results in Section 4.5.3, by observing how samples that are closer to each other, form tighter groups in the latent space, making it easier to separate/distance them from other variants, as well as

by comparing with previous works.

4.4.4 Understanding Evolutionary Changes

In this section, we try to elaborate on the*interpretability* by comprehending what has caused a sample to drift from its neighbors over time. Once our contrastive module has identified groups of evolving variants, it will output a label for each sample, representing the cluster to which it belongs. Given the fact that the model's performance is very tightly coupled with the representations used (i.e., assembly instructions), and while such raw features are very informative for the learning model, there are often not very human-interpretable by themselves.

Therefore, to understand the evolutionary characteristics of evolving IoT malware, we perform a strings-based similarity analysis on the binary samples, which would allow us to attribute the variant changes to more interpretable features. As such, we utilized reverse-engineering techniques and static malware analysis to extract meaningful strings that provide clues about the suspect malware and its functionalities (e.g., attack commands, target devices, malicious payloads, C&C IP addresses, unique strings, etc.). In particular, we use regular expressions to obtain special textual indicators such as IP addresses associated with possible adversarial hosts, and distinctive keywords associated with unique variants or known malicious commands to search for target devices, exploited vulnerabilities and attack operations. Listing 2.2 represents an example of the extracted strings from a malware sample, which is designed to download and execute a malicious file (*bins.sh*) from a possible adversarial C&C server (*http://AnonIP/*).

We also focus on studying the relationships and cross-family agreement between samples, which might be a reflection of reused coding practices among IoT malware families, or a case of mislabeling by anti-virus engines which is quite probable [52, 131].

4.4.5 Evaluation of Instruction Embeddings

We evaluate the quality of our generated cross-architecture code embeddings in terms of their ability to preserve useful semantics information as compared to other baselines. Our qualitative analysis consists of showing that our code embedder can be efficiently used as a semantic search

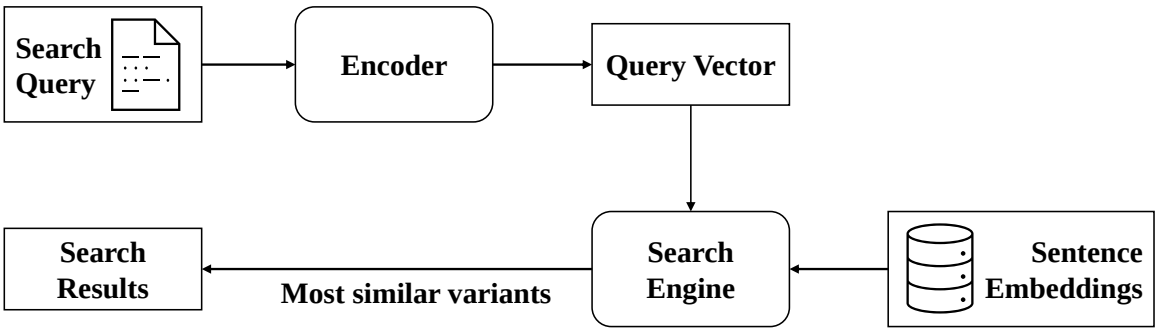


Figure 4.4: Overview of our semantic code search engine.

engine, as shown in Figure 4.4, for finding known variants in our unsupervised dataset with high precision, as well as learning semantic information about syntactically different instructions. In addition, we assess the effect of our well-balanced normalization process in reducing *out-of-vocabulary* instructions when presented with previously unseen and diverse instructions sets. All evaluation results are presented in Section 4.5.5.

4.5 Results

In this section, we examine the impact of the rapid evolution of IoT malware on the performance of classifiers over time, followed by an assessment of the impact of contrastive learning on the detection of in-class drifting IoT malware binaries.

4.5.1 Data Collection

In this paper, we leverage well-known online malware repositories such as VirusShare [51] and VirusTotal [9] along with a specialized IoT honeypot (IoTPOT [6]) to obtain 74,429 IoT malware samples that were detected between 2018 and 2021. Detailed information about the collection process, data cleaning steps, as well as a general description of the dataset is provided in the Background Section 2.2.5.

4.5.2 Observing Concept Drift

In a real-life setting, it is unwise to assume that the data is independent and identically distributed (i.i.d.) [85], therefore, it is most likely that the model will become obsolete when the distribution of incoming data at test-time is different from that of training data. In order to confirm the presence of evolving samples in our dataset, we observe whether such samples impact the accuracy of classifiers. Particularly, we analyse the performance change (i.e., decrease) of a model over time when predicting newly collected data.

To do this, we adopt a sliding time window approach [132] where we begin by splitting the dataset into a training subset T_r with a time window size W_r, and a testing subset T_s with a time window size W_s. For a realistic setting, we enforce the following constraints: **(1)** $W_s > W_r$ to evaluate the long-term performance and robustness to decay of the classifier, **(2)** every window size is split into equal time slots of size z, to allow for a considerable and equal number of samples in each test window $[t_i, t_i + z]$, **(3)** all the samples in T_r must be precedent to the ones in T_s; violating this constraint will bias the evaluation by including future knowledge in the classifier, and **(4)** all the evaluations must assume that labels y_i of the samples $s_i \in T_s$ are unknown, even though we have their labels.

Figure 4.1 captures the performance of a classifier C trained on T_r and tested for each time frame $[t_i, t_i + z]$ of the testing set T_s. We chose a multilayer perceptron (MLP), which is a class of feed-forward artificial neural network (ANN), as our classifier C. Our T_r consists of a random set of samples that have been first scanned by AV engines during the first three months of the year 2018. We first test our classifier on a smaller set of samples which the classifier has not been trained on. Then we test it on samples which have been scanned during subsequent intervals of three months until 2019. As observed in Figure 4.1, the classifier performs well when tested with data that appeared during the same time interval (training period), with an average accuracy above 96%. Subsequently, when we start adding previously-unseen testing samples that have appeared in later months, the overall accuracy significantly drops to below 50%. This is indicative of the changing nature of variants within the same class, and as such, it is necessary to detect such drifting samples.

It is important to note that several mitigation approaches [90, 92] have been proposed to reduce

the performance decay of classifiers when tested on previously-unseen or drifting data, such as retraining on different timestamps, or quarantining drifting samples and retraining the classifier solely on them. Moreover, as our objective is not to test the robustness of classifiers over time nor propose mitigation approaches, we will leave further evaluation of various classifiers and different datasets for future work.

4.5.3 Impact of Contrastive Objective

To examine the impact of using contrastive learning to identify drifting variants within the same class, we first evaluate the quality of the obtained clusters of variants. A good clustering method will produce high quality clusters in which the intra-cluster similarity is low and the inter-cluster similarity is high. The *Silhouette index* is a measure of how similar an object is to its own cluster (*cohesion*) compared to other clusters (*separation*) [133]. A score $s \in [-1,1]$ is calculated for each object, where '1' indicates that this is a perfectly clustered object. Values near 0 indicate overlapping clusters while negative values generally indicate that a sample has been assigned to the wrong cluster.

Given an object x_i of a cluster C, the silhouette score is calculated using the following equation:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{max\{a(x_i), b(x_i)\}}, \tag{4.2}$$

where $a(x_i)$ is the average distance or dissimilarity of an object x_i to all other objects in the same cluster, and $b(x_i)$ is the minimum average dissimilarity of x_i to all other clusters that are not its cluster. The final index is obtained by averaging the scores for all objects in the dataset. Figure 4.5 is a diagrammatic representation of the above-mentioned silhouette coefficient formula.

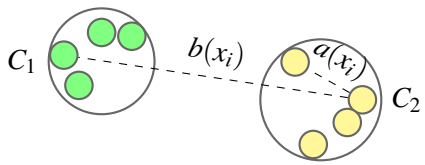
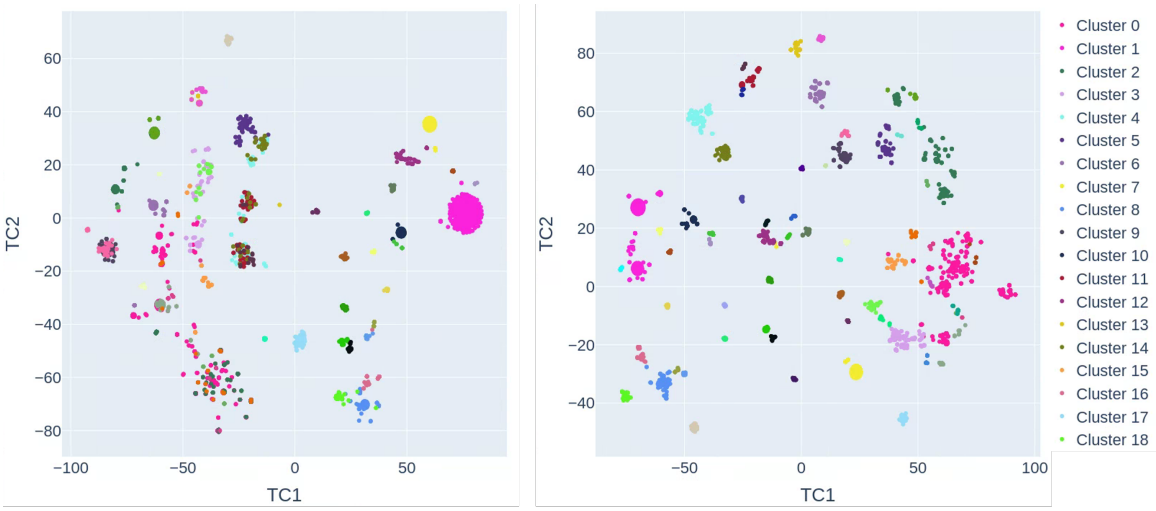


Figure 4.5: A diagrammatic representation of the silhouette coefficient formula $s(x_i)$. C_1 and C_2 are clusters.



(a) Standard **(b) Contrastive**

Figure 4.6: t-SNE visualization of learned representations on 6,000 randomly selected `Mirai` samples with (a) Standard embedding, and (b) Contrastive Embedding.

To illustrate the impact of contrastive learning objective, we compare the learned representations on 6,000 randomly selected `Mirai` samples using Standard and the proposed Contrastive Embedding. As shown in Figure 4.6, we empirically visualize the representations learned by our contrastive module and their distinctive separation in the latent space. We use t-SNE [134], which is a non-linear dimensionality reduction technique for visualizing data in a low 2-dimensional space. We observe that the contrastive loss leads to better variants separation, making it easier to distance samples from others. In fact, the contrastive objective leads to better data clustering than the standard embedding model, with a high average *silhouette score* (about 0.89) for the total number of identified clusters. Moreover, we observe fewer overlapping samples when comparing Figures 4.6.a and 4.6.b and a clearer distinction between them. It is interesting to see the dense pink cluster in Figure 4.6.a further dissected when the contrastive objective is applied in Figure 4.6.b. In fact, our strings-based analysis in Section 4.5.4 confirms the presence of two similar variants in this cluster that only differ by one additional exploit targeting GPON routers. This demonstrastes the effectiveness of EVOLIoT in distinguishing fine-grained modifications in evolved variants.

Comparison with State-Of-Art (CADE [95]). We conduct an experiment to evaluate the performance of the open-sourced CADE [95] when applied to our IoT malware samples. The authors'

Table 4.1: Drifting detection results on the Drebin and IoT malware datasets when comparing CADE with a baseline vanilla autoencoder (AE).

Method	Drebin (Avg)				IoT (Avg)				
	Precision	**Recall**	**F1**	**Norm. Effort**	**Precision**	**Recall**	**F1**	**Norm. Effort**	**Insp. Count**
Vanilla AE	0.63	0.88	0.72	1.48	0.53	0.99	0.69	2.54	22050
CADE	0.96	0.96	0.96	1.00	0.84	0.80	0.82	1.35	9941

objective is to detect and interpret drifting samples from previously unseen malware families by similarly leveraging the power of contrastive learning. By design, CADE uses an auto-encoder augmented with contrastive loss to learn compressed representation of the training data. The first term of their contrastive loss minimizes the reconstruction loss of the auto-encoder while the second term minimizes the Euclidean distance between two samples in the latent space, if they are from the same class. To evaluate the drifting sample detection module, the authors pick one of the families as the *previously unseen family*, while the other families are split into training and testing sets. In this respect, we select three IoT malware families (`Mirai`, `Gafgyt`, `Tsunami`) to form a balanced dataset of their assembly code artifacts, and pick one of the 3 families as the unseen. As such, the unseen family is not available during training and the goal is to correctly identify samples belonging to the hidden family as drifting samples in the testing time. Given a ranked list of detected samples, CADE calculates three evaluation metrics: precision, recall and F_1 score. In addition to these metrics, CADE ranks drifting samples based on their distance to the nearest centroids to focus on those that are furthest away.

As shown in Table 4.1, CADE is compared with a baseline vanilla autoencoder (AE) without contrastive loss. Additionally, we evaluate the performance of CADE when applied to IoT malware samples. For each experiment (choice of previously unseen family), we report the highest F_1 score for each model. The "*inspecting effort*" is a metric that refers to the total number of inspected samples to reach the reported F_1 score, normalized by the number of true drifting samples in the testing set. A higher inspecting effort means that more analysis is required to manually verify if a sample truly belongs to the unseen family in the testing set.

For each evaluation metric, we report the mean value across all settings, as well as the normalized inspecting effort. As summarized in Table 4.1, for the IoT malware dataset, the number of

samples ("inspection count") that need to be validated by security analysts as truly belonging to the previously unseen family is very high (compared to 600 inspected on Drebin by the authors), yet still lower with CADE, which confirms the importance of contrastive learning. Moreover, the obtained results strongly suggest that CADE performs well in most settings (i.e., using the Drebin dataset), but not as well when applied to the IoT malware dataset. This is a limitation of CADE, especially when testing their technique on malware mutations/variants within the same family. As such, CADE is primarily focused on *Type A* concept drift (i.e., introduction of a new class in a multi-class setting), while we address its limitations in identifying drifting samples that are from existing classes (i.e., *Type B*: in-class evolution).

Further Comparison. Several invaluable works have been proposed for clustering IoT malware families using static and dynamic features [46], as well as for dissecting and in-depth studying a singular family such as `Mirai` [2, 7]. Yet, to the best of our knowledge, we are among the first to detect in-class drifting IoT malware binaries and study their mutations over time. In fact, Cozzi et al. [44] proposed a code-level clustering and function similarity solution to draw the lineage of IoT malware families. By design, the authors leveraged a popular off-the-shelf binary diffing tool to perform a detailed code similarity analysis on 93k samples that have appeared between 2015 and 2018, and discovered shared components across different families. While a direct comparison between our works is not feasible, it is clear that their approach is sophisticated and limited by the stripped nature of IoT malware binaries, and by the scalability and direct application of binary diffing tools to IoT binaries.

Moreover, multiple approaches on binary code similarity have been proposed to study malware lineage inference [44,69,75,77,78], however they are only applied in the context in which they were developed and not on Linux-based IoT malware. If they do, they only support the MIPS and ARM architectures. In addition, most proposed solutions compute similarity between binaries from their execution traces [69, 114], which are too-coarse grained for variant identification, or at the function or CFG levels [44,77,80,81,126], which depend on the ability of finding the start of functions. This is inherently difficult in the context of IoT due to the stripped nature of binaries. Cozzi et al. [44] try to circumvent this by propagating known symbols in unstripped binaries to stripped binaries.

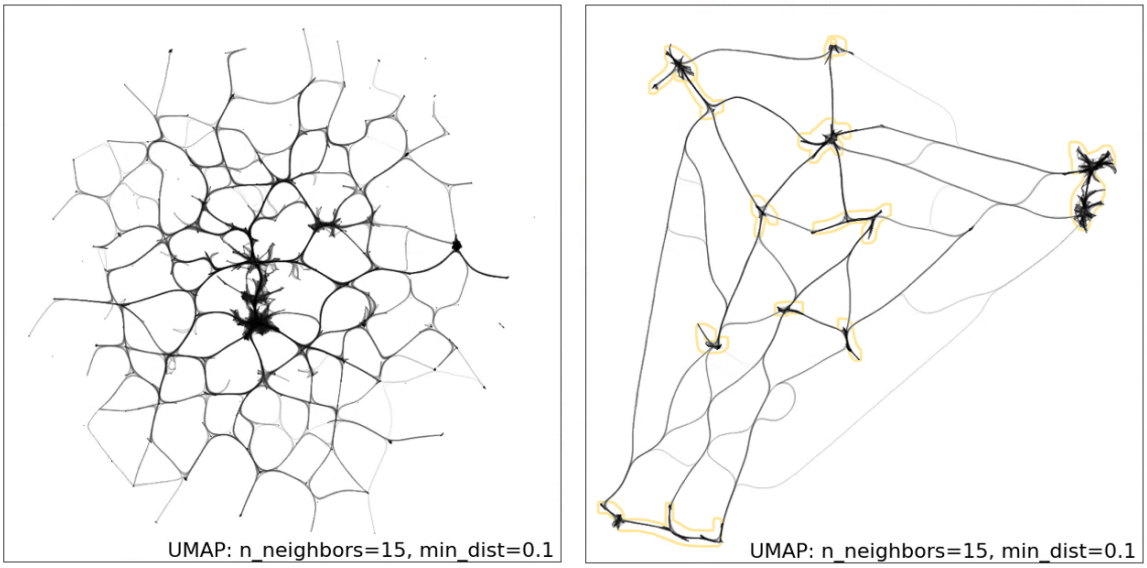
4.5.4 Characterizing Variant Changes

Using our contrastive objective, we identified 44 variants of `Mirai` and 11 variants of `Gafgyt`, as highlighted in Figure 4.7. However, it is still unclear how the same family evolved over time, what makes these variants different, and whether or not samples from different families are connected. To answer these questions, we take a step further by extracting and investigating their *strings*, which will shed more light on the common/differing characteristics of the identified variants.

Given the extracted strings, we perform a string-based similarity analysis on the identified malware samples. As illustrated in figure 4.7, we draw the connectivity plot for 10,000 `Mirai` and 3,000 `Gafgyt` samples. The darker edges are indicators of a high similarity between the samples. By looking at the detection time of the samples, we noticed that the older `Mirai` variants are likely to share more resemblance and appear at dense and more central areas, whereas newer `Mirai` variants are growing apart and appear farther at the edges. In addition, it is clearly observed that the `Mirai` variants are more closely connected, forming more visibly connected regions. On the other hand, `Gafgyt` samples seem to be less connected (lower similarities) and thus, placed further apart. We proceed by understanding these differences and the threats associated with each identified cluster of variants.

Among the three most populated and closely related `Mirai` clusters (#1, #2 and #3), we found 7,124 samples, that were first scanned in 2018, exploiting two vulnerabilities affecting GPON routers. When these two vulnerabilities are used in conjunction, they enable the execution of commands sent by an authorized remote attacker to a vulnerable device. What makes these kindred clusters separate, is their expansion to target a vulnerability in Huawei HG532 routers (#2) and Netgear routers (#3). As such, among samples in Cluster #9, we found 97 attributed to the `Apep` botnet, which aside from dictionary attacks via telnet, also spread through infected Huawei routers, which explains why they are closely-connected to Cluster #2. By investigating further, we also uncovered samples belonging to the `Xjno` variant in the same cluster, using the same command & control URL as the `Apep` variant. This stands to confirm that EVOLIoT identifies fine-grained characteristics shared among different samples.

In addition, we found 1,917 samples spread across Clusters #5 and #11, taking advantage of a



(a) Mirai **(b) Gafgyt**

Figure 4.7: A weighted graph constructed using UMAP [5] representing the connectivity between the strings embeddings of (a) 10,000 `Mirai` and (b) 3,000 `Gafgyt` samples with highlighted nodes that represent 11 variants identified by EVOLIoT (§4.5.4).

ThinkPhP vulnerability which allows them to breach web servers using the PHP frameworks via dictionary attacks on default credentials to gain remote access to them. Among those in Cluster #11, we detected a more recently scanned variant, using a new exploit to infect Huawei devices with malware named after the recent Covid pandemic. Moreover, we found 7 samples belonging to the `Miori` variant in Cluster #8 spreading through a remote code execution in ThinkPHP. They start by contacting other IP addresses using Telnet, while also listening on port 42352 for commands from their C2 servers. Next, they verify whether a targeted device was successfully infected by sending the command "/bin/busybox MIORI". Interestingly, we found one evolved `Miori` sample in Cluster #11 also downloading a new malicious payload named "`corona3.sh`" and killing a list of competing botnets to ensure persistence. As such, by grouping together covid-related samples in Cluster #11, EVOLIoT sheds light on the rapid evolution of IoT malware towards exploiting global events for malware propagation and distribution. Additionally, Cluster #22 contains complementary samples to Cluster #5, belonging to the `Yowai` variant and spreading using the same ThinkPHP exploit, however, we found references to commands for killing competing botnets that might have infected the targeted device. Among their kill list are the names of 58 variants and the majority are

unknown to us. This proves the existence of a multitude of (unknown) IoT variants that attackers are familiar with, which gives them a bigger advantage over the control of the next wave of cyber attacks.

Furthermore, while 70 samples belonging to the `Omni` variant were found in Cluster #1 targeting GPON routers, samples belonging to the same variant were found in a further away cluster, #27. This is because they are leveraging a total of 11 vulnerabilities associated with multiple target devices. While the identified vulnerabilities are publicly known, this is the first variant using all 11 exploits in conjunction. This evolved campaign of `Omni` variants is preventing further infection of infected devices by dropping packets received on 15 different ports using the *iptables* command. Interestingly enough, this variant shares the same IP address for downloading payloads and reporting to the C&C server with 194 `Gafgyt` samples, which explains why they are nearly clustered. This is either a case of mislabeling or an indication that the same bot master is controlling two independent IoT campaigns concurrently. By looking closely at them, we found that they share the same exploits as the `Mirai Omni` variant except for an OS command injection in the UPnP SOAP interface which renders 5 types of D-Link routers vulnerable. However, newer samples of this `Gafgyt` variant detected in 2019 have incorporated a command injection exploit targeting D-Link DSL-2750B routers. As such, while `Mirai` and `Gafgyt` might not share the same codebase, they still exploit and target common devices, using similar attack methods and reporting to the same C2.

Moreover, we performed the same analysis on some `Mirai` samples that have been clustered on the edges, further than the rest, and found that they have been first scanned by VirusTotal between 2020 and 2021. We found 38 samples in Cluster #19 that seem to be spreading by hijacking a vulnerability in digital video recorders (DVR) provided by KGUARD, as well as connecting to their C2 via the Tor-Proxy protocol using 7 different ports. We found embedded URLs with the ".onion" extension. Out of curiosity, we looked at Cluster #29 represented by the singly connected node in Figure 4.7.a, and found indications of the string “`aurora`”. EVOLIoT has pushed it further than other `Mirai` variants because it is spreading using a 0day vulnerability in the Ruijie (NBR700) routers. Interestingly, that same vulnerability is being exploited by an older `Mirai` variant which has appeared two months earlier, which explains the single connection edge. However, the vulnerability exploit payload in the `aurora` variant uses many empty variables with confusing names to distract

security analysts. It additionally has a mechanism to check whether it is running in a sandbox environment, by verifying the path and filename where the sample is located. This is an indication that `Mirai` variants are growing to be more resilient. In addition, while the sample is not packed, a lot of sensitive information are hidden and seem to be encrypted using a different algorithm than `Mirai`'s simple XOR encryption.

4.5.5 Evaluation

To evaluate the quality of our instructions embeddings, we first assess the effectiveness of our model in learning code semantics by finding semantically similar samples in our dataset, compared to other approaches. Then, we evaluate the quality of our cross-architecture instructions in preserving semantics versus syntactics. Finally, we assess the power of normalization in reducing the instructions vocabulary size while capturing code semantics with tokens that hold rich information.

Semantic-Search Engine. To evaluate the quality of our obtained embeddings, our first task aims at finding the most semantically related code from a collection of candidate codes. This would allow us to verify the effectiveness of our model in learning code semantics information. As such, we relied on our strings-based analysis §4.5.4 to build our own ground truth data for evaluation. This dataset consists of a 587 samples of `Mirai` belonging to 6 different variants: `Omni` (70), `Apep` (97), `Yowai` (37), `Satori` (72), `Dark` (91), `Josho` (220). To identify samples that belong to the same variant class, we randomly pick a sample per variant class to encode it using our embedding model and retrieve all other semantically similar code embeddings in our dataset. To do so, we leverage FAISS [135] with Approximate Nearest Neighbor. FAISS uses principal component analysis to reduce the number of dimensions in the vectors, to reduce the computation when comparing a query vector against already embedded vectors. Next, it partitions the data into similar clusters to compare the query vector against these partition/cluster centroids. Once the nearest centroid is found, only full vectors within that centroid are compared to the query, and all others are ignored. Hence, the complexity of the required search area is significantly reduced.

We evaluate the performance of our search engine with varying retrieval thresholds to inspect whether true positives are ranked at the top. We sort the returned results and evaluate each of them in sequence. We collect recall and precision at top-k position ($k = 10$). Recall is the fraction of the

documents that are relevant to the query that are successfully retrieved, while the precision is the fraction of the documents retrieved that are relevant to the user's information need. It is trivial to achieve recall of 100% by returning all documents in response to any query. Therefore, recall alone is not enough but one needs to measure the number of non-relevant documents also, for example by computing the precision. In fact, since each query can have multiple relevant results, we use Mean Average Precision (MAP) as the metric to evaluate the code search on our embeddings dataset. The mean average precision for a set of queries is the mean of the average precision scores for each query, which can be calculated as $MAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$, where Q is the number of queries in the set and $AveP(q)$ is the average precision for a given query q. We use 4 baselines for comparison of the evolution results (Word2vec [136], Doc2vec [137], Code2vec [138], and Bi-LSTM [139]).

We use the following baselines for comparison:

- Word2vec [136] is a popular technique to learn word embeddings using shallow neural networks. The continuous bag-of-words model (CBOW) is a method of word2vec that uses words around a target word as context. In our case, we compare by considering each token (opcode or operand) as word and instructions around each token as its context. DeepBinDiff [84] and Asm2vec [80] both leveraged a variation of word2vec based on CBOW and Paragraph Vector Distributed Memory (PV-DM) respectively, to learn embeddings of assembly functions represented as a control flow graphs (CFGs).

- Doc2vec [137] creates a numeric representation of a document of words, regardless of its length. Distributed Bag of Words (DBOW) is doc2vec algorithm where the paragraph vectors are obtained by training a neural network on the task of predicting a probability distribution of words in a paragraph given a randomly-sampled word from the paragraph. Here, a paragraph represents a code snippet or a basic block of assembly instructions.

- Code2vec [138] is a model that learns distributed representations of code called code embeddings, to evaluate its performance against the task of semantically searching code snippets. It decomposes code fragments to a collection of paths using Abstract Syntax Trees (ASTs) and learns the atomic representation of aggregated paths.

- Bi-LSTM [139] is a bidirectional sequence processing model that consists of two LSTMs:

Table 4.2: Results of the semantic search retrieval using different baselines as code embedders.

Model	Performance (MAP)
Word2vec	0.36
Doc2vec	0.39
Code2vec	0.57
Bi-LSTM	0.74
EVOLIoT	0.89

one taking the input in a forward direction, and the other in a backwards direction. Bi-LSTMs effectively increase the amount of information available and hence improve the context of a word. We treat code simply as sequences of tokens and use the neural machine translation (NMT) baseline (i.e. a 2-layer Bi-LSTM) proposed in [82] for a cross-architecture basic-block comparison.

It is important to note that several other solutions [77, 80, 126] have been proposed to analyze binary code similarity and semantic code retrieval, however, we cannot directly compare them to our approach since they mainly rely on the availability of the source code along with binary functions and/or extracted control flow graphs, which are not considered in our approach.

Besides the above-mentioned baselines that rely on code embeddings, we also manually select features to compare with basic-block feature-based machine learning classifiers such as Gemini [81] and Genius [140]. We compare our embedding model to the SVM classifier using 16 manually selected features (as detailed in Table 4.3 from binary disassembly (e.g., number of instructions, average basic blocks, etc.). We leverage an ELF binary analysis service developed by [6] which evaluates ELF binaries in a multi-architecture sandboxing environment using static and dynamic malware analysis techniques to extract these features.

As shown in Table 4.2, we were able to efficiently retrieve, validate, and label using our semantic search engine, previously unknown variants in our dataset with a mean average precision of 89%. This confirms that our embedding model generate semantically similar/relevant code embeddings, and outperforms different baselines. Moreover, our model outperforms the SVM classifier trained on the manually selected features, and achieves much higher AUC values, as illustrated in Figure 4.8. This is because manually selected features cause significant information loss in terms of the

Table 4.3: List of manually extracted features from binaries disassembly using Padawan ELF tool [6].

Feature	Description
average_bytes_function	Average size in bytes of a function
average_basic_blocks	Average number of basic blocks with respect to functions
average_cyclomatic_complexity	Average cyclomatic complexity with respect to functions
average_location	Average lines of code with respect to functions
branch_instruction	Number of branch instructions
bytes_function	Total size in bytes of the functions
call_instructions	Numberof call instructions
function_location	Percentage of instructions belonging to functions
indirect_branch_instruction	Number of indirect branch instructions
max_basic_blocks	Max basic blocks
max_cyclomatic_complexity	Max cyclomatic complexity
num_funcs	Number of functions detected
percent_load_covered	Percentage of covered load segment
percent_text_covered	Percentage of covered text section
syscall_instructions	Number of syscall instructions

contained instructions and the dependencies between these instructions, while our model precisely encodes and preserves the block semantics across different variants.

Cross-Architecture Instruction Embeddings. In this section, we present our results from qualitatively analyzing the instruction embeddings for different architectures. Zuo et al. [82] have shown that instructions compiled for the same architecture cluster together while those compiled from a different architecture cluster far from each other. This is due to the syntactic variations that an architecture introduces, which creates further challenges in the context of IoT due to the diverse nature of cross-platform devices.

Our objective is to evaluate the quality of our cross-architecture instruction embedding model, by picking variants that have been clustered together for their semantic relationship, and analyzing their target architectures. For our embedding model to be effective, two semantically similar yet syntactically different binaries should still be clustered closely in the space. As such, we first use t-SNE [122] to plot the instruction embeddings in a two-dimensional plane. As shown in Figure 4.9, our analysis of clusters 1, 3, and 14 show that semantically related samples are clustered together even if they have different target instruction architectures. In fact, cluster 1 contains semantically

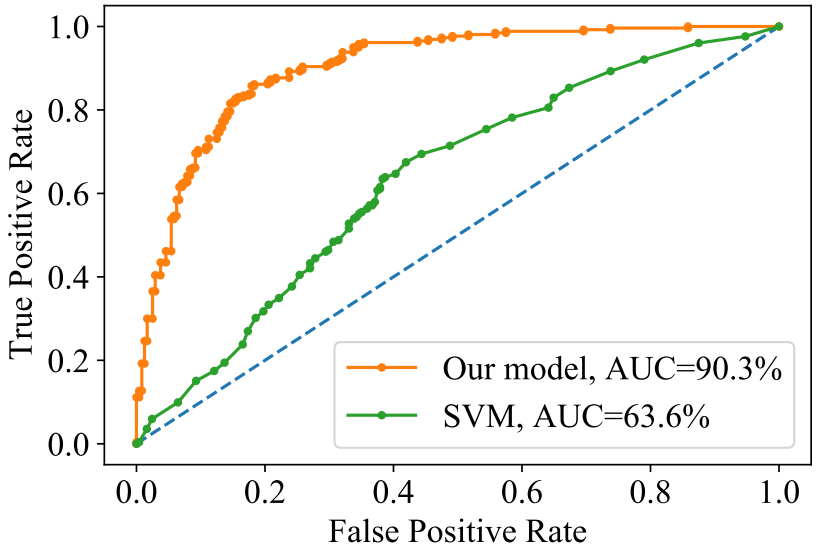


Figure 4.8: Comparing ROC and AUC performance results.

similar samples targeting a variety of CPU architectures, while clusters 3 and 14 are only reserved to the top two most popular architectures among IoT devices. Such variability is explained by evolving samples that are expanding the pool of potential devices which can be compromised.

Effectiveness of Well-balanced Normalization. A preprocessing normalization step is applied to avoid an out-of-vocabulary problem, while preserving code semantics instructions with tokens that hold rich information. To evaluate the impact of instructions normalization, we seek to understand whether the instructions vocabulary size is affected with or without pre-processing. As such, we disassembled 4,385 `Mirai` variants that have been scanned in 2019 and counted the total number of assembly instructions: 80,299,331. Then, we divided the corpus in 10 equal sized parts to see how the vocabulary size grows in terms of the percentage of analyzed corpus. As clearly observed in Figure 4.10, the vocabulary size grows significantly and relatively fast without pre-processing, while remains relatively small when pre-processing is applied.

As shown by the analysis results, EVOLIoT can be leveraged as a semantic search engine for finding semantically similar variants while outperforming previous approaches, in addition to preserving semantics in cross-architecture embeddings and reducing out-of-vocabulary instructions.

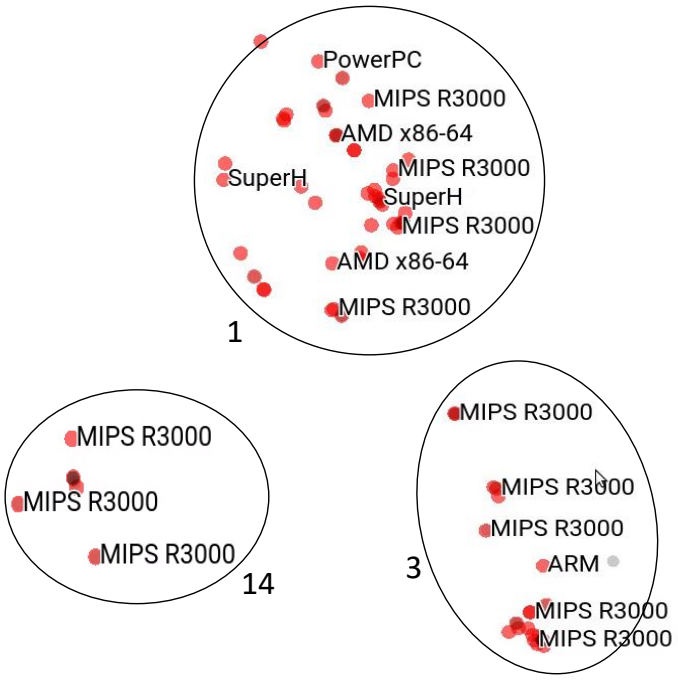


Figure 4.9: Visualization of syntactically different yet semantically similar embeddings belonging to clusters 1, 3, and 14.

4.6 Limitations and Future Work

This work has a number of limitations, which may hamper the generalizability and the validity of the reported findings. For instance, we rely on the fact that IoT malware samples are still largely unobfuscated as compared to generic malware. However, we were unable to extract useful strings from around 23% of the analyzed samples. Despite that, our analysis showed that the majority of these samples did not employ sophisticated obfuscation, thus, can be de-obfuscated using off-the-shelf tools (e.g., UPX [100]). Furthermore, due to the lack of fine-grained IoT malware labels and lack of well-designed ground truth datasets for evaluation, we built our own ground truth to evaluate the proposed semantic search engine (§4.5.5). As such, while the dataset might be relatively small and not representative, it represents a reliable dataset since the samples were carefully examined and labelled manually through our strings-based analysis. For future work, we intend to address the above mentioned limitations by investigating different techniques for malware deobfuscation and unpacking to account for a more representative sample of IoT malware while extending the work to non-IoT malware space, which observes more obfuscation. Further, we will combine our

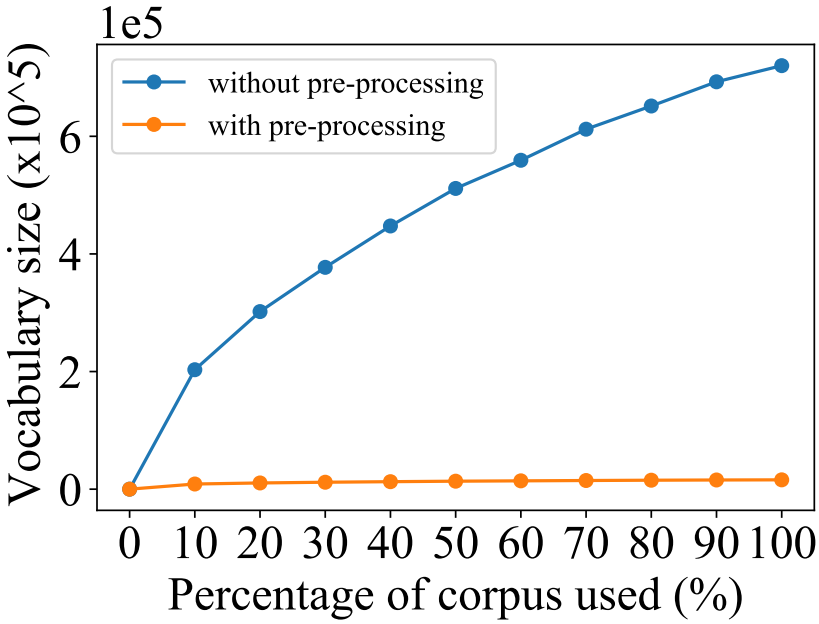


Figure 4.10: Visualization of the growth of the vocabulary size when the corpus size increases.

manual analysis with information obtained from online threat repositories to obtain a more representative ground truth that can improve our evaluation outcomes. Finally, the unveiling of diverse IoT malware variants inspires us to study in future work the competition and coordination among IoT botnets in the wild, which is still underexplored in literature.

4.7 Summary and Concluding Remarks

In this work, we present a robust, accurate, and semantic-aware assembly instructions representation generator, EVOLIoT, which leverages the evolution of IoT binaries as effective augmentation strategy for contrastive learning. Our approach achieves both efficiency and accuracy for cross-architecture assembly instructions search without relying on any expensive (e.g., CFG) or manually selected features. Additionally, we addressed the problem of in-class concept drift by detecting evolving IoT malware variants and interpreting the reason behind their drift. Further, we comprehensively evaluate the effectiveness and robustness of our proposed approach with a large corpus of IoT malware data. Our findings shed light on the evolving IoT threat landscape characterized by the ever-lasting `Mirai` variant, which is spreading by using new undisclosed vulnerabilities, persisting

by killing other bots and looking out for sandbox environments, encrypting its communication using Tor proxies, and incorporating encryption algorithms.

Chapter 5

Conclusion and Future Work

Despite their benefits, the insecurity of the Internet of Things (IoT) devices has turned them into attractive targets for orchestrated large-scale cyber attacks (e.g., DDoS). The rising number of malware-driven cyber attacks in recent years poses a major threat to the IoT realm. Moreover, the complex relationship and similarities in terms of code reuse among IoT malware variants bring several challenges for malware analysis including labeling, provenance, triage, lineage analysis, as well as family and authorship attribution.

We dedicate this thesis to overcome the limitations of malware family attribution and inter-family classification due to the evolutionary nature of IoT malware. Considering the peculiarities of the IoT paradigm and the various challenges associated with it, we successfully achieved the latter by employing deep-learning based approaches and static malware analysis techniques. In particular, we addressed the lack of empirical data about IoT malware by obtaining malware binaries collected by an IoT-tailored honeypot.

Consequently, we are among the first to introduce a holistic, multi-level approach for analyzing malware by combining the benefits of static malware analysis with deep learning classification techniques. While our analysis results indicate the effectiveness of our proposed classification model for attributing malware samples to known families, we also leverage the multi-level classifier to predict the labels of unknown malware samples that have not been detected/labeled by major AV vendors.

In addition to building an effective learning based-malware classifier against the spread of IoT malware, we extend this thesis to build a better understanding of emerging IoT malware variants.

Specifically, we identify when the model shows signs of aging by which it fails to effectively recognize new variants. Furthermore, we utilize reverse-engineering and static malware analysis techniques to build a cross-architecture code-based analysis that can capture a binary's malicious and evolutionary essence. Particularly, we leverage the power of contrastive learning and the cutting-edge BERT Transformer model to deeply infer and compare the underlying fine-grained code semantics of a binary without the need for expensive target labels. This is essential in a security application where access to labeled data (e.g., malware) is challenging. Subsequently, we leverage our interpretable strings-based analysis to address the evolution of IoT malware binaries by correlating the observed drifting variants into groups with common malicious objectives and functional differences.

As for future work, we aim to utilize our drift detector and explanation module as building blocks to preparing a robust learning-based application for the open-world environment. Moreover, our multi-level classifier is designed based on the assumption that the training set does not contain mislabeled samples (or adversarial samples). Hence, we defer to future work robustifying our system against poisoned malicious labels. Furthermore, the evolving nature of IoT malware sheds light on the growing number of obfuscation and encryption in the IoT malware space, which motivates to investigate in the future different techniques for malware deobfuscation and unpacking to account for any hidden information that might hinder our analysis. Finally, the unveiling of evolving/drifting IoT malware variants inspires us to put some effort in studying the competition and coordination among IoT botnets in the wild, to understand the ongoing battle over the Internet of Things fought by `Mirai` and its many siblings.

Bibliography

[1] M. S. C. Release, "https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released/," 2016.

[2] 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 *USENIX Security*, 2017.

[3] S. U. Pascual. (2007, October) Layout of an elf file. Retrieved from https://commons.wikimedia.org/wiki/File:Elf-layout--en.svg.

[4] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[5] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[6] 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.

[7] P.-A. Vervier and Y. Shen, "Before toasters rise up: A view into the emerging iot threat landscape," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 556–576.

[8] 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)*, Washington, D.C, USA, 2015.

[9] "VirusTotal." [Online]. Available: https://www.virustotal.com/

[10] M. Dib, S. Torabi, E. Bou-Harb, and C. Assi, "A multi-dimensional deep learning framework for iot malware classification and family attribution," *IEEE Transactions on Network and Service Management*, 2021.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[12] C. Cimpanu. (2018, March) Hajime Botnet Makes a Comeback With Massive Scan for MikroTik Routers. Retrieved from https://www.bleepingcomputer.com/news/security/hajime-botnet-makes-a-comeback-with-massive-scan-for-mikrotik-routers/.

[13] J. Vijayan. (2018, January) Satori Botnet Malware Now Can Infect Even More IoT Devices. Retrieved from https://www.darkreading.com/vulnerabilities-threats/satori-botnet-malware-now-can-infect-even-more-iot-devices.

[14] Z. Whittaker. (2017, April) Homeland Security warns of 'BrickerBot' malware that destroys unsecured internet-connected devices. Retrieved from https://www.zdnet.com/article/homeland-security-warns-of-brickerbot-malware-that-destroys-unsecured-internet-connected-devices/.

[15] P. Muncaster. (2014, October) Massive Qbot Botnet strikes 500,000 Machines Through WordPress. Retrieved from https://www.infosecurity-magazine.com/news/massive-qbot-strikes-500000-pcs/.

[16] A. Anubhav. (2017, July) Agile QBot Variant Adds NbotLoader Netgear Bug in Its New Update. Retrieved from https://blog.newskysecurity.com/agile-122bf2f4e2f3.

[17] ——. (2018, January) Masuta : Satori CreatorsâĂŹ Second Botnet Weaponizes A New Router Exploit. Retrieved from https://blog.newskysecurity.com/masuta-satori-creators-second-botnet-weaponizes-a-new-router-exploit-2ddc51cc52a7.

[18] 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.

[19] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[20] H. P. Enterprise, "Internet of things research study," *Internet of Things Research Study*, 2015.

[21] L. Markowsky and G. Markowsky, "Scanning for vulnerable devices in the internet of things," in *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, vol. 1. IEEE, 2015, pp. 463–467.

[22] 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*, 2010, pp. 97–106.

[23] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *2019 IEEE symposium on security and privacy (sp)*. IEEE, 2019, pp. 1362–1380.

[24] S. Farahani, *ZigBee wireless networks and transceivers*. newnes, 2011.

[25] V. Sachidananda, S. Siboni, A. Shabtai, J. Toh, S. Bhairav, and Y. Elovici, "Let the cat out of the bag: A holistic approach towards security analysis of the internet of things," in *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security*, 2017, pp. 3–10.

[26] F. Loi, A. Sivanathan, H. H. Gharakheili, A. Radford, and V. Sivaraman, "Systematically evaluating security and privacy for consumer iot devices," in *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, 2017, pp. 1–6.

[27] C. Cimpanu. (2017, February) A Hacker Just Pwned Over 150,000 Printers Left Exposed Online . Retrieved from https://www.bleepingcomputer.com/news/security/a-hacker-just-pwned-over-150-000-printers-left-exposed-online/.

[28] A. Tekeoglu and A. Ş. Tosun, "A testbed for security and privacy analysis of iot devices," in *2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2016, pp. 343–348.

[29] D. Lodge. (2016, January) Steal your Wi-Fi key from your doorbell? IoT WTF! Retrieved from https://www.pentestpartners.com/security-blog/steal-your-wi-fi-key-from-your-doorbell-iot-wtf/.

[30] S. Morgenroth. (2017, April) How I Hacked my Smart TV from My Bed via a Command Injection. Retrieved from https://www.netsparker.com/blog/web-security/hacking-smart-tv-command-injection/.

[31] G. Hernandez, O. Arias, D. Buentello, and Y. Jin, "Smart nest thermostat: A smart spy in your home," *Black Hat USA*, no. 2015, 2014.

[32] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 95–110.

[33] C. Konstantinou and M. Maniatakos, "Impact of firmware modification attacks on power systems field devices," in *2015 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2015, pp. 283–288.

[34] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: A novel honeypot for revealing current iot threats," *Journal of Information Processing*, vol. 24, no. 3, pp. 522–533, 2016.

[35] J. D. Guarnizo, A. Tambe, S. S. Bhunia, M. Ochoa, N. O. Tippenhauer, A. Shabtai, and Y. Elovici, "Siphon: Towards scalable high-interaction physical honeypots," in *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, 2017, pp. 57–68.

[36] U. D. Gandhi, P. M. Kumar, R. Varatharajan, G. Manogaran, R. Sundarasekar, and S. Kadu, "Hiotpot: surveillance on iot devices against recent threats," *Wireless personal communications*, vol. 103, no. 2, pp. 1179–1194, 2018.

[37] S. Dowling, M. Schukat, and H. Melvin, "A zigbee honeypot to assess iot cyberattack behaviour," in *2017 28th Irish signals and systems conference (ISSC)*. IEEE, 2017, pp. 1–6.

[38] T. Luo, Z. Xu, X. Jin, Y. Jia, and X. Ouyang, "Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices," *Black Hat*, pp. 1–11, 2017.

[39] A. Costin and J. Zaddach, "Iot malware: comprehensive survey, analysis framework and case studies," *BlackHat USA*, 2018.

[40] H. Griffioen and C. Doerr, "Examining mirai's battle over the internet of things," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 743–756.

[41] S. Torabi, E. Bou-Harb, C. Assi, E. B. Karbab, A. Boukhtouta, and M. Debbabi, "Inferring and investigating iot-generated scanning campaigns targeting a large network telescope," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[42] D. Lee. (2014, September) Shellshock: 'Deadly serious' new vulnerability found. Retrieved from https://www.bbc.com/news/technology-29361794.

[43] L. Foundation. (2001, April) Elf and abi standards. Retrieved from https://refspecs.linuxfoundation.org/.

[44] 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.

[45] A. Darki, M. Faloutsos, N. Abu-Ghazaleh, M. Sridharan *et al.*, "Idapro for iot malware analysis?" in *12th USENIX Workshop on Cyber Security Experimentation and Test (CSET19)*, 2019.

[46] S. Torabi, M. Dib, E. Bou-Harb, C. Assi, and M. Debbabi, "A strings-based similarity analysis approach for characterizing iot malware and inferring their underlying relationships," *IEEE Networking Letters*, vol. 3, no. 3, pp. 161–165, 2021.

[47] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, "Malware dynamic analysis evasion techniques: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–28, 2019.

[48] D. Gibert, C. Mateu, and J. Planes, "HYDRA: A Multimodal Deep Learning Framework for Malware Classification," *Computers & Security*, p. 101873, 2020.

[49] C. Jindal, C. Salls, H. Aghakhani, K. Long, C. Kruegel, and G. Vigna, "Neurlux: Dynamic Malware Analysis Without Feature Engineering," in *Proceedings of the 35th Annual Computer Security Applications Conference*, San Juan, PR, USA, 2019, pp. 444–455.

[50] 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*, 2020, pp. 169–182.

[51] "VirusShare." [Online]. Available: https://virusshare.com/

[52] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *International symposium on research in attacks, intrusions, and defenses*. Springer, 2016, pp. 230–253.

[53] M. S. Pour, A. Mangino, K. Friday, M. Rathbun, E. Bou-Harb, F. Iqbal, S. Samtani, J. Crichigno, and N. Ghani, "On data-driven curation, learning, and analysis for inferring evolving internet-of-things (iot) botnets in the wild," *Computers & Security*, vol. 91, p. 101707, 2020.

[54] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware Images: Visualization and Automatic Classification," in *Proceedings of the 8th Int. Symposium on Visualization for Cyber Security*, 2011, pp. 1–7.

[55] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification," in *Proceedings of the Sixth ACM Conf. on Data and Application Security and Privacy*, 2016, pp. 183–194.

[56] T. Beppler, M. Botacin, F. J. Ceschin, L. E. Oliveira, and A. Grégio, "L (a) ying in (test) bed," in *International Conference on Information Security*. Springer, 2019, pp. 381–401.

[57] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Using convolutional neural networks for classification of malware represented as images," *J. of Computer Virology and Hacking Techniques*, vol. 15, no. 1, pp. 15–28, 2019.

[58] 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. 2. IEEE, 2018, pp. 664–669.

[59] R. Tian, L. Batten, R. Islam, and S. Versteeg, "An Automated Classification System Based on the Strings of Trojan and Virus Families," in *4th Int. Conf. on Malicious and Unwanted Software (MALWARE)*. Montreal, QC, Canada: IEEE, 2009, pp. 23–30.

[60] 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)*. IEEE, 2018, pp. 1–9.

[61] H.-T. Nguyen, Q.-D. Ngo, and V.-H. Le, "Iot botnet detection approach based on psi graph and dgcnn classifier," in *2018 IEEE International Conference on Information Communication and Signal Processing (ICICSP)*. IEEE, 2018, pp. 118–122.

[62] Y. Zhang, Q. Huang, X. Ma, Z. Yang, and J. Jiang, "Using Multi-features and Ensemble Learning Method for Imbalanced Malware Classification," in *IEEE Trustcom/BigDataSE/ISPA*, Tianjin, China, 2016, pp. 965–973.

[63] M. Mays, N. Drabinsky, and S. Brandle, "Feature Selection for Malware Classification," in *MAICS*, Fort Wayne, IN, USA, 2017, pp. 165–170.

[64] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *J. of Network and Computer Applications*, vol. 36, no. 2, pp. 646–656, 2013.

[65] Y. Zhao, W. Cui, S. Geng, B. Bo, Y. Feng, and W. Zhang, "A malware detection method of code texture visualization based on an improved faster rcnn combining transfer learning," *IEEE Access*, vol. 8, pp. 166 630–166 641, 2020.

[66] G. Bendiab, S. Shiaeles, A. Alruban, and N. Kolokotronis, "Iot malware network traffic classification using visual representation and deep learning," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 444–449.

[67] J. Yan, G. Yan, and D. Jin, "Classifying malware represented as control flow graphs using deep graph convolutional neural network," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 52–63.

[68] H. Alasmary, A. Abusnaina, R. Jang, M. Abuhamad, A. Anwar, D. NYANG, and D. Mohaisen, "Soteria: Detecting adversarial examples in control flow graph-based malware classifiers," in *40th IEEE International Conference on Distributed Computing Systems, ICDCS*, 2020, pp. 1296–1305.

[69] J. Jang, M. Woo, and D. Brumley, "Towards automatic software lineage inference," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 81–96.

[70] I. U. Haq and J. Caballero, "A survey of binary code similarity," *arXiv preprint arXiv:1909.11424*, 2019.

[71] D. Hull, "Computer viruses: naming and classification, part ii," in *Virus Bulletin Conference*, 1995.

[72] L. A. Goldberg, P. W. Goldberg, C. A. Phillips, and G. B. Sorkin, "Constructing computer virus phylogenies," *Journal of Algorithms*, vol. 26, no. 1, pp. 188–208, 1998.

[73] T. Dumitras and I. Neamtiu, "Experimental challenges in cyber security: A story of provenance and lineage for malware." *CSET*, vol. 11, pp. 2011–9, 2011.

[74] M. E. Karim, A. Walenstein, A. Lakhotia, and L. Parida, "Malware phylogeny generation using permutations of code," *Journal in Computer Virology*, vol. 1, no. 1, pp. 13–23, 2005.

[75] M. Lindorfer, A. Di Federico, F. Maggi, P. M. Comparetti, and S. Zanero, "Lines of malicious code: Insights into the malicious software industry," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 349–358.

[76] A. Calleja, J. Tapiador, and J. Caballero, "The malsource dataset: Quantifying complexity and code reuse in malware development," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 12, pp. 3175–3190, 2018.

[77] H. Huang, A. M. Youssef, and M. Debbabi, "Binsequence: fast, accurate and scalable binary code reuse detection," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 155–166.

[78] J. Ming, D. Xu, and D. Wu, "Memoized semantics-based binary diffing with application to malware lineage inference," in *IFIP International Information Security and Privacy Conference*. Springer, 2015, pp. 416–430.

[79] W. M. Khoo, A. Mycroft, and R. Anderson, "Rendezvous: A search engine for binary code," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 329–338.

[80] S. H. Ding, B. C. Fung, and P. Charland, "Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization," in *IEEE Symp. on Security and Privacy (SP)*. IEEE, 2019, pp. 472–489.

[81] 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 *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 363–376.

[82] F. Zuo, X. Li, P. Young, L. Luo, Q. Zeng, and Z. Zhang, "Neural machine translation inspired binary code similarity comparison beyond function pairs," *arXiv preprint arXiv:1808.04706*, 2018.

[83] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding," *arXiv preprint arXiv:1703.03130*, 2017.

[84] Y. Duan, X. Li, J. Wang, and H. Yin, "Deepbindiff: Learning program-wide code representations for binary diffing," in *Network and Distributed System Security Symposium*, 2020.

[85] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[86] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Brazilian symposium on artificial intelligence*. Springer, 2004, pp. 286–295.

[87] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 2007, pp. 443–448.

[88] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," *arXiv preprint arXiv:1610.02136*, 2016.

[89] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2008, pp. 108–125.

[90] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 625–642.

[91] A. Deo, S. K. Dash, G. Suarez-Tangil, V. Vovk, and L. Cavallaro, "Prescience: Probabilistic guidance on the retraining conundrum for malware detection," in *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, 2016, pp. 71–82.

[92] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "{TESSERACT}: Eliminating experimental bias in malware classification across space and time," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 729–746.

[93] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[94] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," *arXiv preprint arXiv:1612.04433*, 2016.

[95] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "Cade: Detecting and explaining concept drift samples for security applications," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[96] M. Nawir, A. Amir, N. Yaakob, and O. B. Lynn, "Internet of Things (IoT): Taxonomy of Security Attacks," in *3rd Int. Conf. on Electronic Design (ICED)*. Phuket, Thailand: IEEE, 2016, pp. 321–326.

[97] T. Brosch and M. Morgenstern, "Runtime packers: The hidden problem," *Black Hat USA*, 2006.

[98] I. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," in *2010 Int. Conf. on Broadband, Wireless Computing, Communication and Applications*. Los Alamitos, CA, USA: IEEE, 2010, pp. 297–300.

[99] FireEye Labs Obfuscated String Solver, "FLOSS," *Retrieved from https://github.com/fireeye/flare-floss*, 2016.

[100] M. F. Oberhumer, "UPX the Ultimate Packer for eXecutables," *http://upx. sourceforge. net/*, 2004.

[101] M. Raghu, C. Zhang, J. Kleinberg, and S. Bengio, "Transfusion: Understanding transfer learning for medical imaging," *arXiv preprint arXiv:1902.07208*, 2019.

[102] N. Thai-Nghe, Z. Gantner, and L. Schmidt-Thieme, "Cost-Sensitive Learning Methods for Imbalanced Data," in *The 2010 Int. joint conference on neural networks (IJCNN)*. Piscataway, NJ, USA: IEEE, 2010, pp. 1–8.

[103] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, 2016.

[104] Autonomio, "Talos," *Retrieved from http://github.com/autonomio/talos*, 2019.

[105] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[106] R. M. Haralick, K. Shanmugam, and I. H. Dinstein, "Textural features for image classification," *IEEE Transactions on systems, man, and cybernetics*, no. 6, pp. 610–621, 1973.

[107] S. Gatlan, "Exposed Docker APIs Abused by DDoS, Cryptojacking Botnet Malware," June 2019. [Online]. Available: https://www.bleepingcomputer.com/news/security/exposed-docker-apis-abused-by-ddos-cryptojacking-botnet-malware/

[108] N. Ben Said, F. Biondi, V. Bontchev, O. Decourbe, T. Given-Wilson, A. Legay, and J. Quilbeuf, "Detection of Mirai by Syntactic and Semantic Analysis," Nov. 2017, preprint. [Online]. Available: https://hal.inria.fr/hal-01629040

[109] N. Provos *et al.*, "A virtual honeypot framework." in *USENIX Security Symposium*, vol. 173, no. 2004, 2004, pp. 1–14.

[110] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, 2011, pp. 43–58.

[111] S. Edwards and I. Profetis, "Hajime: Analysis of a decentralized internet worm for iot devices," *Rapidity Networks*, vol. 16, pp. 1–18, 2016.

[112] V. J., "Satori Botnet Malware Now Can Infect Even More IoT Devices," *Retrieved from https://www.darkreading.com/vulnerabilities---threats/satori-botnet-malware-now-can-infect-even-more-iot-devices/d/d-id/1330875*, 2018.

[113] Q.-D. Ngo, H.-T. Nguyen, V.-H. Le, and D.-H. Nguyen, "A survey of iot malware and detection methods based on static features," *ICT Express*, vol. 6, no. 4, pp. 280–286, 2020.

[114] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering." in *NDSS*, vol. 9. Citeseer, 2009, pp. 8–11.

[115] T. Gao, X. Yao, and D. Chen, "Simcse: Simple contrastive learning of sentence embeddings," *arXiv preprint arXiv:2104.08821*, 2021.

[116] F. Ceschin, M. Botacin, H. M. Gomes, L. S. Oliveira, and A. Grégio, "Shallow security: On the creation of adversarial variants to evade machine learning-based malware detectors," in *Proceedings of the 3rd Reversing and Offensive-oriented Trends Symposium*, 2019, pp. 1–9.

[117] D. M. dos Reis, P. Flach, S. Matwin, and G. Batista, "Fast unsupervised online drift detection using incremental kolmogorov-smirnov test," in *Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2016, pp. 1545–1554.

[118] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9729–9738.

[119] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[120] K. K. Yang, Z. Wu, C. N. Bedbrook, and F. H. Arnold, "Learned protein embeddings for machine learning," *Bioinformatics*, vol. 34, no. 15, pp. 2642–2648, 2018.

[121] A. X. Lu, H. Zhang, M. Ghassemi, and A. M. Moses, "Self-supervised contrastive learning of protein representations by mutual information maximization," *BioRxiv*, 2020.

[122] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[123] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv:1409.0473*, 2014.

[124] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[125] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE, 2007, pp. 421–430.

[126] L. Massarelli, G. A. Di Luna, F. Petroni, R. Baldoni, and L. Querzoni, "Safe: Self-attentive function embeddings for binary similarity," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2019, pp. 309–329.

[127] S. Eschweiler, K. Yakdan, and E. Gerhards-Padilla, "discovre: Efficient cross-architecture identification of bugs in binary code." in *NDSS*, vol. 52, 2016, pp. 58–79.

[128] Hex-Rays. (2005) IDA Pro Disassembler. https://www.hexrays.com/products/ida/.

[129] Z. Wu, S. Wang, J. Gu, M. Khabsa, F. Sun, and H. Ma, "Clear: Contrastive learning for sentence representation," *arXiv preprint arXiv:2012.15466*, 2020.

[130] Y. Meng, C. Xiong, P. Bajaj, S. Tiwary, P. Bennett, J. Han, and X. Song, "Coco-lm: Correcting and contrasting text sequences for language model pretraining," *arXiv preprint arXiv:2102.08473*, 2021.

[131] A. Mohaisen and O. Alrawi, "Av-meter: An evaluation of antivirus scans and labels," in *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 2014, pp. 112–131.

[132] J. Costa, C. Silva, M. Antunes, and B. Ribeiro, "Concept drift awareness in twitter streams," in *2014 13th International Conference on Machine Learning and Applications*. IEEE, 2014, pp. 294–299.

[133] F. Iglesias, T. Zseby, and A. Zimek, "Absolute cluster validity," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 9, pp. 2096–2112, 2019.

[134] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[135] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *arXiv preprint arXiv:1702.08734*, 2017.

[136] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[137] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*. PMLR, 2014, pp. 1188–1196.

[138] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, 2019.

[139] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv:1503.00075*, 2015.

[140] Q. Feng, R. Zhou, C. Xu, Y. Cheng, B. Testa, and H. Yin, "Scalable graph-based bug search for firmware images," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 480–491.