Graph-Based IoT Malware Family Classification

by

Nastaran Mahmoudyar

Bachelor of Science in Computer Engineering, IUST, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s):	Ali A. Ghorbani, PhD, Faculty of Computer Science
	Arash Habibi Lashkari, PhD, Faculty of Computer Science
Examining Board:	Rongxing Lu, PhD, Faculty of Computer Science
	KaliKinkar Mandal, PhD, Faculty of Computer Science
	Donglei Du, PhD, Faculty of Management

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

January 2021

Abstract

Internet of Things malware has become one of the main cyber-threats nowadays. There is no comprehensive study in a feature-based manner for IoT malware detection approaches to the best of our knowledge. Moreover, the studies show that there is a lack of IoT malware family classification system. This thesis attempts to bridge these gaps by proposing a feature-based IoT malware taxonomy and a graph-based IoT malware family classification framework by combining the FCGs and fuzzy hashes. We introduce the Aggregated Weighted Graph (AWGH) of Hashes, representing each IoT malware family's structure. We use IDA Pro [60] for generating the FCGs, *ssdeep* [3] for computing the fuzzy hashes, and Python for developing the fully automated framework. To evaluate the system's effectiveness, we use the VirusTotal dataset [4] and provide a comparative analysis with different IoT malware regarding their CPU architectures (MIPS, ARM, i386, PowerPC, and AMD64). The results show the effectiveness of our framework.

Dedication

I dedicate my dissertation work to my dear family and my adored husband with love and eternal appreciation. A special feeling of gratitude to my loving parents, Nasrin & Hamid, whose without their endless love and unconditional supports, I could not jump towards my desires. Their words of encouragement and push for tenacity always ring in my ears to move me forward. Also, I dedicate this thesis to my sister, Mobina, who showed that distance is not an obstacle to love.

Last but not least, to my beloved husband, Hamid Azimy, who has faith in my dreams more than me and always supports me to fly with an extra pair of wings and his power of love and wisdom.

Acknowledgements

I would like to express my sincere appreciation to both of my supervisors, Dr. Ali A. Ghorbani and Dr. Arash Habibi Lashkari, for their guidance, suggestions, and kind support throughout the process of my master's study and completing this dissertation.

Table of Contents

List of Tables

List of Figures

Abbreviations

IoT	Internet of Things
RFID	Radio Frequency Identification
CFG	Control Flow Graph
FCG	Function Call Graph
ELF	Executable and Linkable Format
ML	Machine Learning
GA-PID	Genetic Algorithm Placement of IoT Device
CW	Confidence Weighted Learning
CPVT	Cyber-Physical Voice privacy Theft
DNS	Domain Name Service
LR	Logistic Regression
SVM	Support Vector Machines
RF	Random Forest
DT	Decision Tree
CNN	Convolutional Neural Network
KNN	k-Nearest Neighbors
GED	Graph Edit Distance
PSI	Printable String Information
IG	Information Gain
MFP	Maximal Frequent Patterns
DDoS	Distributed Denial-of-Service
CTPH	Context Triggered Piecewise Hashing
GDL	Graph Description Language
NCCIC	National Cybersecurity Communications Integration Center

Chapter 1

Introduction

1.1 Introduction

The Internet of Things (IoT) [6] is the network of devices that include sensors, software, and actuators to send/receive data to/from other connected wireless devices through standard communication ways including Radio Frequency Identification (RFID), Zigbee, WiFi, Bluetooth, and 3G/4G/5G. IoT devices are equipped with different CPU architectures other than x86/x64, including MIPS, i386, ARM, and Motorola 68020. The IoT network is similar to the traditional Internet. In other words, a considerable number of smart devices such as connected appliances, smart home security systems, wireless inventory trackers, and Smart factory equipment are always connected to the Internet for transmitting the data from one place to another over the Internet. Imagine billions of sensors and connected IoT devices that

work for every possible industry of our society, including manufacturing, energy distribution, smart cities, smart agriculture, smart buildings, and smart medical equipment to improve user experiences [18].

The popularity of using IoT devices increases in people's daily lives due to making everything integrated, connected, easy access to the data, and linked to everything. Moreover, IoT devices have a positive impact on our environment by using less energy and better sustainability. Therefore, IoT vision is getting more understood to bring convenience and efficiency to human living. Indeed, IoT devices are sorts of micro versions of traditional embedded devices. There are lots of conventional attacks that target IoT devices as well. Likewise, the Distributed Denial-of-Service (DDoS) attack is a well-known attack in IoT objects as much as traditional systems.

IoT devices' main characteristics, such as always being connected to the Internet and lacking proper protection mechanisms, encourage adversaries to target them [22]. IoT devices are exposed because of many security problems, such as weak passwords, backdoors, and various vulnerabilities. IoT devices' vulnerabilities such as buffer overflow and authentication bypass create a proper environment for executing the malicious code by attackers.

The increase in IoT devices has raised the number of malicious software (malware) for the IoT environment. Based on the [10], the number of IoT malware has doubled in the last year. Thus, the IoT malware subject is one of the most dangerous threats in cybersecurity. Certainly, proposing effective IoT malware detection mechanisms is essential for providing safety

and security in IoT network communications or dealing with massive IoT data.

1.2 IoT Malware Definition

In the beginning, we represent a specific definition of IoT malware to show the particular characteristics of IoT malware recording to the hardware or software limitations. The more accurate the description, the better and more practical solutions we can offer to avoid adversaries' attempts. Determining a particular definition for IoT malware based on the nature of IoT helps the researchers to focus on studying differences and similarities between IoT malware families. For Proposing realistic solutions, extensive studies in the IoT malware field are required.

As we discussed in the previous section, the Internet of Things (IoT) refers to the network of connected embedded devices to save and transfer data using sensors. First of all, we go in-depth of the IoT objects to know the main cores of IoT objects:

- Sensors: Sensors are responsible for the data collection part in the IoT objects. In other words, sensors gather habitual information about what exactly should be collected for further analysis or monitoring tasks. Sensors capture external information and change the data to the signals that computers can understand.

- Actuators: Actuators operate to control the IoT data or act on the

data. Indeed, actuators play an essential role in maintaining or moving a required section of a specific system. In other words, actuators convert electricity to some mechanical actions.

IoT devices' two main components are the real point of interest for adversaries because attackers could access data through them. By compromising the sensors or actuators, attackers can steal sensitive information or take control of the object.

After understanding IoT objects' different security aspects, we can provide a better definition of IoT malware. IoT malware is a piece of malicious software that takes control of the targeted system with or without the user's knowledge. Usually, the main goal of infecting IoT devices with malware is a preparation step before a major attack in the future rather than to commit minor data theft. Here are some of the significant cybersecurity risks that IoT malware may cause: Access to sensitive data, sabotage, botnet, and ransomware.

A significant number of IoT malware will be gone right after rebooting the system. Moreover, the size of IoT malicious files are typically lighter than other types of malware. IoT malware samples are usually supporting different types of CPU architectures despite Personal Computer (PC) malware samples that mostly target a specific machine type. IoT malware samples use different ways to gain access to the victim system. The most popular initial access methods for IoT adversaries include:

- Phishing: Attackers operate phishing methods, especially for android users, due to making a bridge from the connected phone to the IoT objects.

- Backdoor or open port: Most vendors leave some backdoor on the products to get easy access for further updates or other types of required actions.

The IoT malware domain is relatively new compared to classical types of malware. Although malware detection has been a well-studied topic for windows-related malware, it brings unique challenges to the IoT domain. IoT devices are usually limited in resources such as computational memory and energy resources. Besides, IoT objects typically are equipped with low-profile processors. Indeed, applying resource exhausting malware analysis tools [65] and classical approaches against PC malware are not proper for IoT devices [10]. So, it brings up a new security challenge.

It is now necessary to answer the following questions to distinguish between IoT and PC malware: 1) What is so special about IoT malware? 2) What is the difference between IoT malware and PC malware?

Costin *et al.* (2018) determine some of the differences between IoT malware and PC malware in the following Table 1.1.

The differences between the PC malware and IoT malware in behavioural manners depend on the specific type of malware individually. Besides, the attacking techniques that malware samples use for targeting victim devices

Table 1.1: IoT Malware Vs. PC Malware [22]

	PC	IoT
Platform heterogeneity	low	high
Malware family plurality	high	low
Detection on the system	easy	hard
In-vivo analysis	easy	very hard
Sandbox execution	easy	hard
Removal	medium	hard to impossible
Vulnerability assessment	medium	very hard

play an important role in behavioural differences. Other than differences in the source code and the architecture, there are differences in the operations. For example, Mirai is a prevalent malware to compromise IoT devices. In traditional systems, the infection phase, which creates Mirai botnets, is different from making the IoT Mirai zombies in IoT networks [16].

In Traditional Mirai malware, the infection phase operates directly by bots. However, in the IoT perspective, the Mirai zombie task is just scanning the whole network to find the vulnerable devices to notify the Command & Control (C&C) server. Indeed, IoT Mirai zombies only inform the C&C server of the potential zombies, and the infection occurs by the communications with the C&C server.

1.3 IoT Malware & Lack of Security

The number of IoT devices which will be compromised by malware is increasing. The reason behind this is that infecting IoT connected devices is much simpler than conventional systems for the following reasons [42]:

- IoT devices are always on and connected to the Internet.
- IoT devices are less secure than traditional systems—for example, lack of knowledge in IoT devices' users to keep the platform up to date. Moreover, the lack of practical security mechanisms is because of memory and energy limitations in IoT devices.
- Conventional security methods in computers such as cryptography techniques are not appropriate for providing security IoT devices.
- Most IoT devices have a common weakness, which is using default credentials or not selecting strong username-passwords.
- IoT manufacturers keep an open port to access the devices remotely for further updates.
- Customers do not use firewalls to connect the IoT devices to the Internet. So, unfortunately, most of the IoT devices connect to the Internet directly.

In recent years, due to IoT architecture breaches and low quality of IoT software codes [36], adversaries target IoT devices by the use of malware and

make lots of DDoS attacks as well. Nowadays, malware is the most severe problem for IoT devices due to easy installation on the victim device without the owners' awareness [65]. For presenting detection approaches in the IoT malware domain, the researchers face the following main challenges [10]:

1. Most of the time, IoT malware families support different types of CPU architectures, such as MIPS, i386, ARM, and Motorola 68020.

2. Since IoT resources are limited, the proposed detection mechanism should be efficient in energy and computing resource consumption. Previous approaches for traditional malware are not appropriate for IoT because most of the conventional methods against PC malware are too complicated for IoT devices in reality.

1.4 IoT Malware Detection or Classification

Admittedly, the IoT malware domain is not as well-studied as malware on Android and Windows platforms. The majority of IoT malware samples support running over Linux-based systems. Executable and Linkable Format (ELF) is a standard file format for executable files and object code. IoT attackers develop malicious files in ELF format. After executing the malware code on the compromised system, the malware takes absolute control of the victim system through the infected malware by making communication between the C&C server and the target [8].

IoT malware detection and classification are essential due to making defensive mechanisms to fight against malicious software. Thus, understanding the exact behaviour, characteristics, and structure of the IoT malware is vital for further analysis and proposing security mechanisms. In this regard, we are analyzing various types of IoT malware families to determine their family structure for presenting a classification framework.

From the perspective of security, one of the prominent threats to any system is the zero-day attack. Making an effective system to classify unknown malware based on the family types is fundamental for a quick and proper security response to the threat. Nowadays, it is vital in the IoT field to have adequate studies and proposed security mechanisms because of the lack of security that we mentioned before and the growing number of connected devices in industry and private lives.

On the other hand, IoT weaknesses are not threats to just IoT systems. Adversaries can penetrate a non-IoT system through IoT devices' vulnerabilities. The reason is that nowadays, mobile phones and PCs are connected to other smart devices, including IoT objects. Thus, a lack of IoT security can be dangerous for different types of devices as well. Accordingly, attackers use PCs or mobile phones as bridges to reach the main IoT targets. Indeed, the attackers design the attacks by taking advantage of PCs and mobile phones to compromise the connected IoT devices.

Building a classification model in IoT that can distinguish between various IoT malware families is substantial to address the mentioned shortcoming.

Grouping the malware samples into the appointed families reflects the similarities and distinguishable differences for applying the appropriate security mechanisms against the attack. Toward IoT malware classification, we build a graph-based framework to answer the mentioned problems.

1.5 Summary of Contributions

The main purpose of this research is to classify IoT malware samples of each particular family. In summary, the following are the contributions of this thesis:

1. Classifying IoT malware families based on the proposed family graph (Con_1);
2. Providing a novel taxonomy of IoT malware detection approaches (Con_2);
3. Surveying different groups of features for IoT malware detection in previous studies (Con_3);
4. Creating an Aggregated Weighted Graph of Hashes (AWGH) for each IoT malware family which represents the family structure to determine the similarity and differences between each incoming IoT malware and all known IoT malware families (Con_4);
5. Proposing a new malware family classification approach in IoT using the fuzzy hash technique (Con_5);

6. Creating a fully automated system based on the call graphs (FCGs) to classify IoT malware into the belonged families regarding the type of the CPU architecture (Con_6).

1.6 Thesis Organization

The rest of the thesis is organized as follows.

- Chapter 2: *Background* discusses the required definitions for understanding the previous and current studies, including the graph representing, malware analysis, fuzzy hashing, and graph comparison algorithms.

- Chapter 3: *Literature Review & Proposed Taxonomy* reviews previous related work in different IoT malware approaches. Moreover, Chapter 3 presents the taxonomy of IoT malware detection approaches, including the motivation, the overview of taxonomy, and the classification of features in previous studies.

- Chapter 4: *Proposed Framework* provides a general overview of the proposed framework, followed by a description of each phase. It explains how an input binary sample is passed through three phases to be classified into the appointed families.

- Chapter 5: *Implementation* describes criteria of the proposed framework. This chapter introduces the main components and classes imple-

mented in our framework and lists their main functions.

- Chapter 6: *Experiments & Results* represents the results of the research implementation. This chapter reports the dataset, metrics, and results of the proposed framework.

- Chapter 7: *Conclusion & Future Work* concludes the thesis with the contributions' summary, challenges and limitation of conducting the research, and some remarks on the future work.

Chapter 2

Background

2.1 Overview

In this chapter, we provide the needed prerequisite information for understanding the current studies in this field. Indeed, we have a quick review of the background definitions, including malware analysis, graph representation, fuzzy hashing, and graph comparison algorithms.

First, we look at the different types of malware analysis. Second, we review the graph representation techniques to answer the following questions: What is the graph representation? What information can be extracted from the graph representation of a binary?

And then, we review fuzzy hashing to know its definition and functionality. After that, we discuss the usage of fuzzy hashing in malware analysis. At the end of this section, we describe the graph similarity methods to measure

the graph similarity?

2.1.1 Static & Dynamic Malware Analysis

Increasing the number of IoT devices has caused IoT-related malicious software to compromise the connected smart devices as available new targets. Recently, IoT botnets have compromised several commercial devices like IP-camera, router, and smart TV as the latest victims [51].

The first step is to find the malware's actual behaviour and structure for analyzing a new malware sample. Analyzing a malware sample reveals useful information, such as network behaviour, memory functionality, and obfuscation mechanisms. Indeed, the analysis results can clarify the valuable vulnerabilities and potential risks for the target system [39].

Malware analysis has been done in two following manners [63]:

- **Static**: Static analysis does not need to run the malware to discover that sample's information. In this case, mostly the extracted data includes n-gram of opcodes, n-gram of strings, imported and exported libraries, and functions. Although this technique is straightforward, obfuscated malware can thwart the analysis [46] due to altering parts of the code; meanwhile, it is running.

- **Dynamic**: Dynamic analysis is running the malware in an isolated environment like a sandbox to precisely capture the real behaviour of the sample. So if the malware uses obfuscation mechanisms, the dynamic

analysis determines more accurate information than static analysis. Nevertheless, a few malware samples use anti-virtualization techniques and anti-debugging methods to distinguish between the real environment and virtual scenarios [26]. In that case, understanding the actual behaviour of the malware is a mystery even in dynamic analysis.

2.1.2 Graph Representations

Graph representation displays the properties of the binary and binary relationships. On the other hand, a graph representation of an object describes the relationships of edges and nodes. Also, the main functionality of the executable object reveals by graph representations. In that regard, graph-based solutions can play an essential role in malware classification approaches.
In the following parts, we will explain Function Call Graphs (FCGs) and Control Flow Graphs (CFGs), which are different graph representations.

2.1.2.1 Function Call Graphs (FCGs)

FCGs show functions' relationships in an executable file. FCGs represent how functions call each other to be executed as a program. The graph-based features from FCGs represent the executable files' structural aspects, unlike other features such as n-gram of opcodes [33].
Besides, call graphs show the interaction between functions, which means call graphs display relationships of functions' calls.

FCGs can be generated by static or dynamic analysis depending on the types of approaches.

- **Static**: A static FCG describes every possible run of the binary. So, static FCGs approximately represent the relationships of every single call. Static FCGs consist of all likely calls. In other words, by running the code in a real environment, part of the function calls that exist in the code may never happen. The main negative point of using the static technique is that sometimes the malware's actual running code is much different from the code that has been analyzed.

  In these cases, the malware code can be packed or obfuscated. In general, when the code is changed because of obfuscation techniques, static graph representations cannot describe the malicious programs' real structure and characteristics.

- **Dynamic**: A dynamic FCG provides an accurate representation of a single run of the malware binary. Because the binary is executed for the analysis and it is obvious the exact real pattern of calling functions in a program even if it is obfuscated. On the other hand, dynamic FCGs show the actual structure and function calls of a program.

Figure 2.1 represents the complete overview of FCG for one of the IoT malware in the "Gafgyt" family, while Figure 2.2 shows a zoomed piece of Figure 2.1.

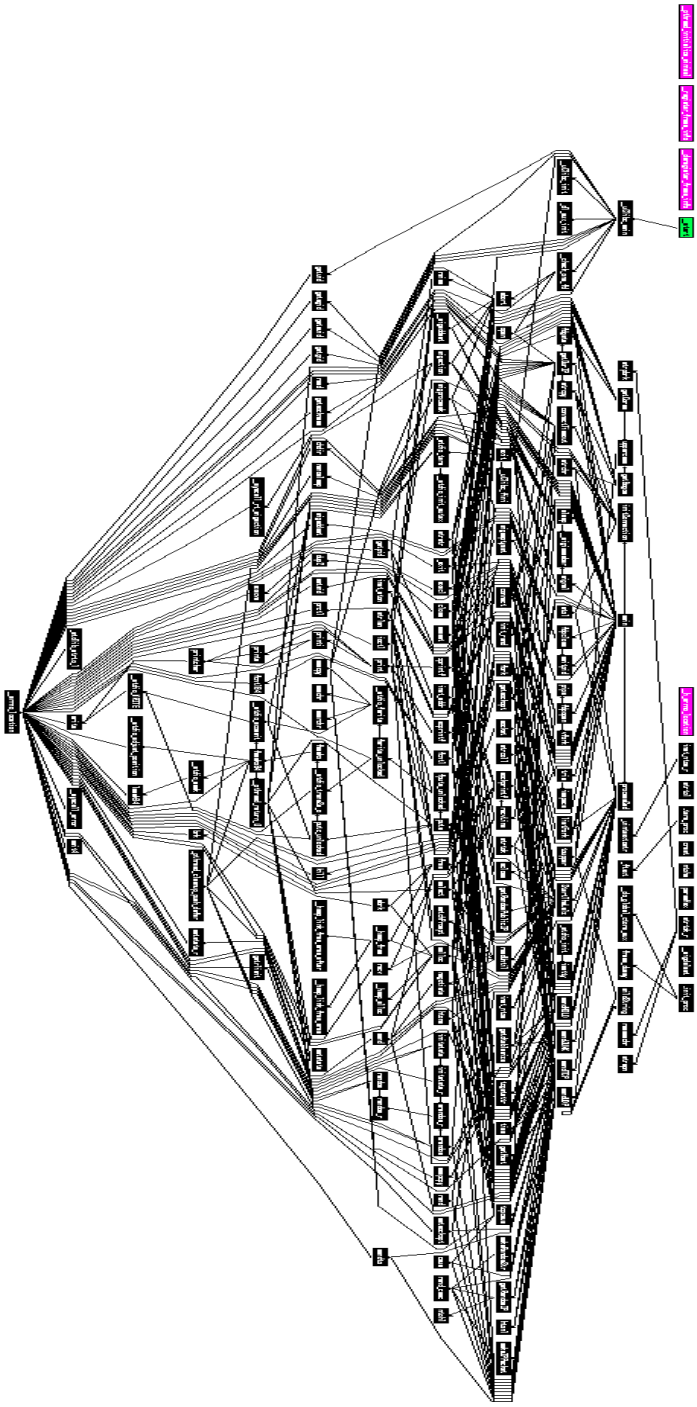


Figure 2.1: The complete FCG for one Gafgyt Sample

In function call graphs representations, each node describes one function of the malicious code, and the whole connected edges display the relationships of all the functions to each other.

As you can see in Figure 2.1, FCG is a complicated graph with many nodes and edges, which is impossible to read the details at once glimpse.

So Figure 2.2 represents a small zoomed-in part of the complete FCG Figure 2.1 to show the inside of the nodes (black boxes) and details of the real relationships of functions. Each black box displays a single function of the selected malware sample.

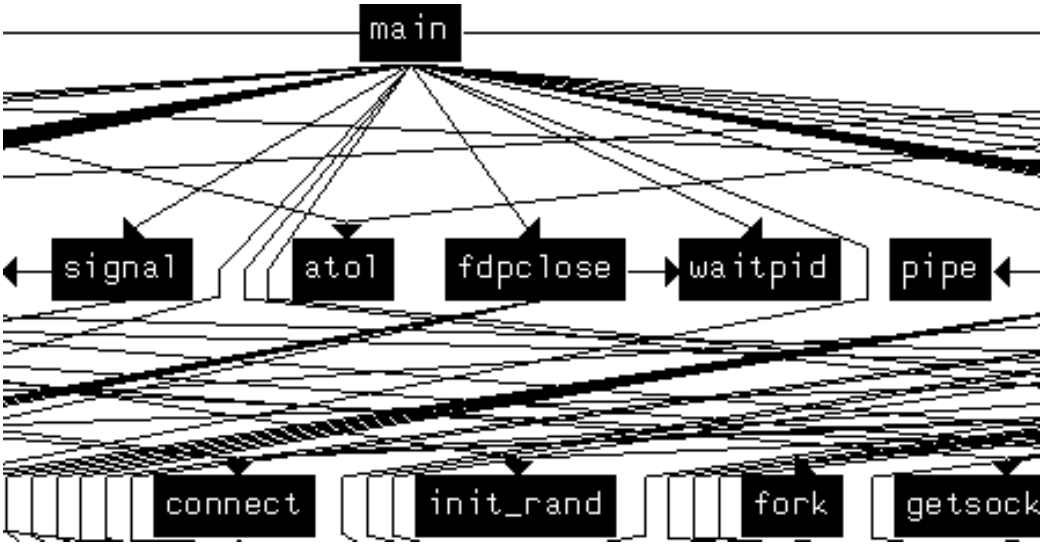


Figure 2.2: A small part of the FCG for one Gafgyt Sample

Parts of the previous works use the graph of Printable String Information (PSI) instead of the original FCG. PSI graph is a simple graph compared to CFGs, and it is generated based on the printable strings information in the

binary code of the malicious software.

The PSI-graph represents the exact relationship of the functions the same as FCGs, but the PSI-graph contains parts of the binary code that have remarkable "printable strings" not all of it. Since the whole malware's activities on the system are not insidious and harmful, the code contains the regular activities as well [50]. PSI-graphs make the process much easier and faster because of working on a smaller version of FCGs with important information for malware detection.

PSI-graphs mostly apply for detection approaches when the goal is distinguishing between benign and malicious software. The PSI-graph is mostly not proper for malware family classification because malicious software contains similar patterns in malicious code manner, even in different families. However, PSI-graph might be useful for differentiating between benign and malicious software due to distinguishable patterns.

2.1.2.2 Control Flow Graphs (CFGs)

CFGs show the executable path of a malware binary. In CFGs, each node is a statement of the code, and each edge represents the control flow of the statements. CFGs use for metamorphic malware detection because CFGs can display the functionality of an executable file. CFGs can be generated by static or dynamic analysis like FCGs. CFGs suffer from the same limitations and weaknesses of FCGs in the static approaches as well. Figure 2.4 represents the complete overview of the Control Flow (CFG) graph for an

IoT malware sample in the “Gafgyt” family.

In addition, Figure 2.3 shows a zoomed piece of Figure 2.4 for better understanding of the represented graph.

In CFGs, each node describes one statement of the malware program, and the whole CFGs represents the total flow of the binary.

Figure 2.3 is a small zoomed-in part of Figure 3.7 to show the inside of the black boxes.

Each block consists of a piece of code, and the flow represents the relationships of all blocks.

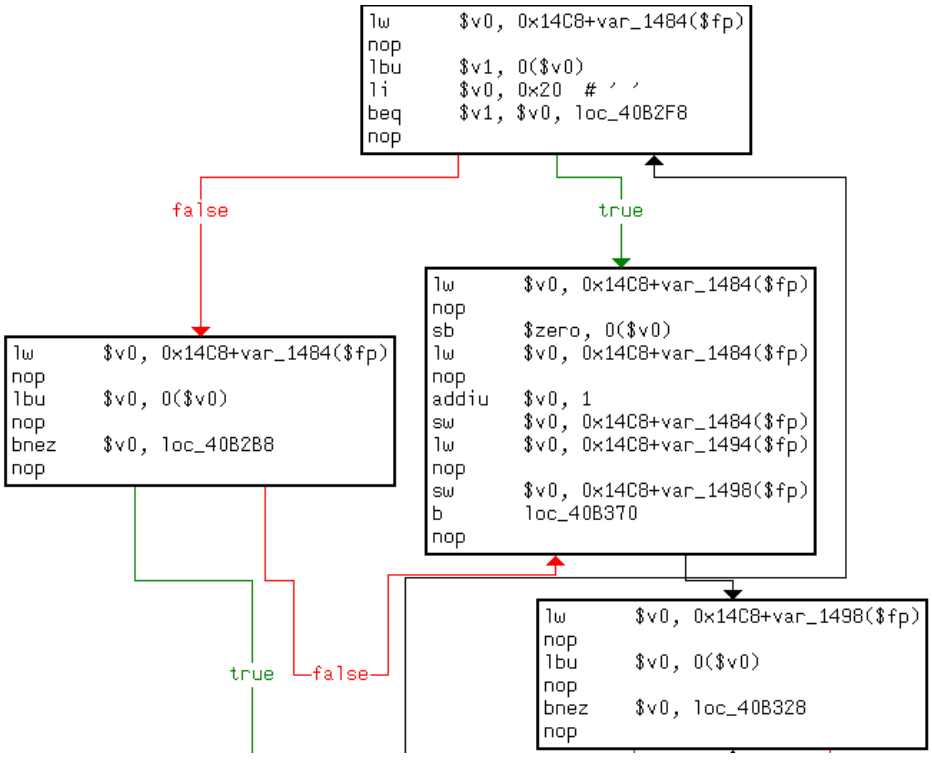


Figure 2.3: A small part of the CFG for one Gafgyt Sample

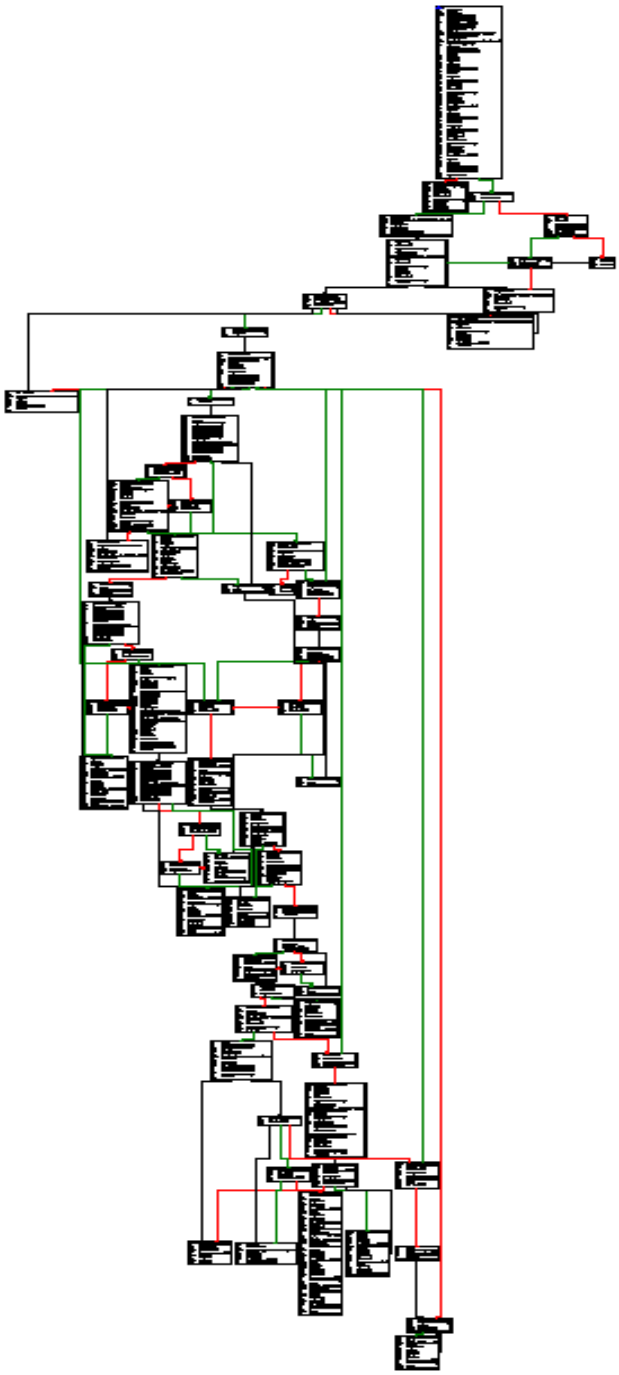


Figure 2.4: The complete CFG for one Gafgyt Sample

2.1.3 Fuzzy Hashing

Context Triggered Piecewise Hashing (CTPH) is a compression function that can measure the similarity between the files. It is also known as fuzzy hashing algorithms, fuzzy hash function, similarity preserving hash functions, or similarity digest.

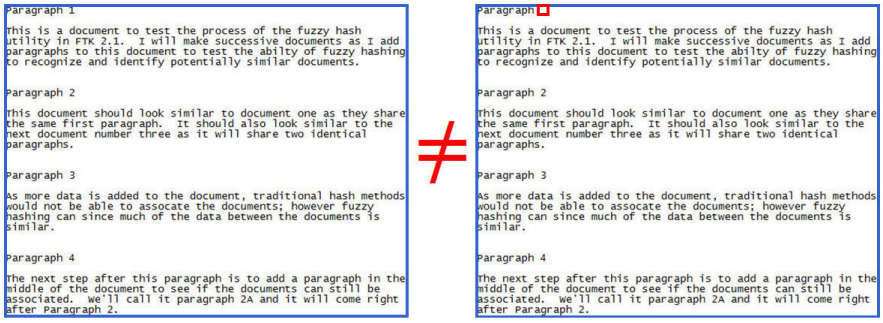
Using the fuzzy hash is similar to the fuzzy logic search. In other words, forensic investigators use fuzzy hash techniques for finding the similarity between the requirement files when the exactness of the documents is not essential. The typical hashing techniques like MD5 has been used for finding the modification in the files. Certainly, fuzzy hashing is useful to determine the similarity between files, not the exactness.

Although traditional cryptography hash algorithms like MD5 have been used to determine the equivalence between files, the fuzzy hash function compares fingerprints of input files to measure the similarity. So the big difference between the usual hashing techniques like MD5 and fuzzy hashing algorithms is that traditional cryptography hash algorithms are used to compare digital files' exactness.

For understanding the fuzzy hash, getting familiar with the other kinds of hashes techniques, including cryptographic, rolling, and context hashing, is necessary [34].

- **Cryptographic Hash**: As we mentioned above, the use of traditional hashing is for finding the exactness in the searching requirement files to

determine the shreds of evidence in some forensics cases. Changing just one bit of data leads to a total huge change of the hash. So, presenting the associations within the documents is not possible by cryptographic hashes. Figure 2.5 provides an example of how traditional hashing works with a minor data change.



File 1 | File 2

File 1 MD5 Hash = aa060a95f2732612f80dd80e5f33b0f6
File 2 MD5 Hash = 290423c55ee6f9f48c45e6e02d126420

Figure 2.5: Traditional Cryptographic Hash Changes with a Slight Change of Data [34]

- **Rolling Hash**: Rolling hash performs based on the cutting data in several pieces. Indeed, the rolling hash engine is responsible for creating the "trigger" that helps the segmentation. Once the trigger is defined, all segments will be created, and for each segment, the cryptographic hash will be calculated. So, the rolling hash consists of the sequences of traditional hashes of the data, which is calculated in several pieces. Segmentation in the rolling hash is defined by the trigger, based on the

size of the file.

- **Context Triggered Piecewise Hash (CTPH)**: As mentioned before, CTPH is working upon finding the similarity among the documents and not the exact duplicates of them. CTPH segments are generated according to the context of data, not the size. Then, the final hash is displayed using the segmented hashes to compare documents. Figure 2.6 shows the four trigger of the file by red boxes. So between each trigger, the traditional hash is calculated and archived.

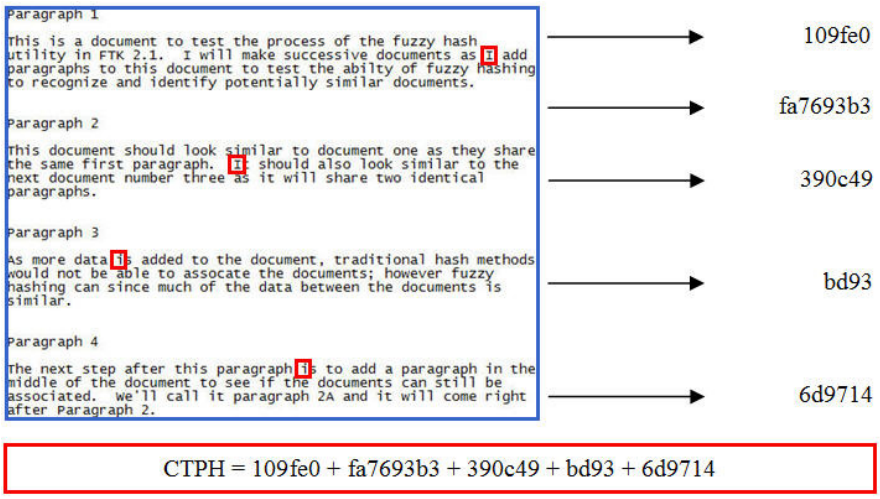


Figure 2.6: Context Triggered Piecewise Hash Example [34]

Recently, fuzzy hashing has been used to classify similar malware samples. Fuzzy hashing is appropriate for grouping malware samples based on their signatures. For instance, different malware families can use the same C&C or have similar behaviour or activities. The traditional hashes of various digital

files are completely different because small changes in files make significant differences in hashes. However, the fuzzy hashes indicate the similarity of files and the number of changes in the hash's value to compare the files. Thus, fuzzy hashing techniques are being used as a customized distance function for the clustering like PC malware samples in [43].

Authors [43] use fuzzy hashing and clustering methods. First, Li *et al.* (2015) compute each malware binary fingerprint. After that, the distance of each pair is calculated. Based on the threshold that has been set, all samples are clustered in groups, and the final output is different malware families for Windows-based malware samples. In [45], authors work on Android malware samples to generate feature vectors for Android families' classification. In this work, Mirzaei *et al.* (2019) uses a combination of FCGs and Fuzzy hashing to create a weighted hash graph rely on ensembles of API methods. Furthermore, as an interesting example in geneticists, fuzzy hashing techniques use for comparison between the gene sequence of an unknown microorganism and a known genome [38]. This thesis calculates the fuzzy hashes of all samples' functions by *ssdeep*. Therefore, we can define all samples' functions' similarity and exactness by computing the fuzzy hashes. IoT malware binaries may contain two types of functions in their codes. So, malware functions can be either of the following types:

- Local functions: Written by the developer of the malware.
- External functions: External functions includes system and library

function calls. External functions are standard and unique in their assigned names.

Depending on the types of functions, calculating the fuzzy hashes is different. If it is a local function, fuzzy hashes of the whole body of the function are required. If not, the only name of the external functions reflects the uniqueness of the contents.

2.1.3.1 SSDEEP

ssdeep is one of the main tools for detecting malware modification by matching the homologies in inputs [4]. *ssdeep* performs based on the sequences of matching bytes in the identical arrangement. However, some bytes among the sequences can have disparate content or even length [1].

ssdeep is still used by VirusTotal and the National Cybersecurity Communications Integration Center (NCCIC) for identification endeavors [4]. The high speed of the *ssdeep* makes it popular compared to other similar tools like TLSH.

At first, Jesse Kornblum created *ssdeep* according to the "SpamSum" program by Dr. Andrew Trigdell, which is for spam message detection. Since improving the program by Jesse Kornblum, *ssdeep* has been used for detecting small changes of files to detect infected files by malware. *ssdeep* works on most operating systems, including Windows, Ubuntu, Debian, and other *nix platforms. Besides, the Linux-based source code for *ssdeep* is publicly available at GitHub [3].

2.1.4 Graph Comparison

In this part, we make a quick summary of the existing popular approaches for calculating the similarity between the two required graphs. After that, we examine the types of graph matching approaches and subgraph matching methods.

In the end, we discuss whether each of these methods is useful for our proposed framework or not. Besides, Chapter 4 and Chapter 5 discuss what will be used in our proposed method.

2.1.4.1 Graph Similarity

In this part, we review the proposed approaches for graph similarity to see whether the available techniques are appropriate for our suggested framework or not.

All presented techniques for finding the similarity between the graphs can be arranged into three main groups [40]:

- Edit distance/graph isomorphism: Two graphs with the exact number of edges connected in the same way called isomorphic. And if two graphs are isomorphic, they are similar. Likewise, if a graph is isomorphic to a subgraph of the other, they are similar. Also, both graphs have subgraphs that are isomorphic to each other.

- Feature extraction: Similar graphs share some properties like degree distribution, diameter, eigenvalues. The similarity between the ex-

tracting features represents the graphs are similar.

- Iterative methods: "Two nodes are similar if their neighbourhoods are also similar". For doing so, exchange similarity scores of each node measured in each iteration, and it is done if convergence is reached.

2.1.4.2 Graph Matching

The main goal of graph matching is to determine the similarity between the two graphs' nodes and edges. In other words, graph matching techniques find similar structures in both graphs that are matched. Regarding it, the similarity score of the two graphs can be calculated.
Graph matching techniques can be grouped into two following categories:

- Exact Matching: Exact matching algorithms indicate whether two graphs or even part of them are identical based on structure and labels. This technique usually is applied when the goal is searching within a bigger substructure for a smaller one [17].

- Inexact Matching: When two graphs are almost the same, and the differences are some missing nodes or edges in one of them, inexact matching can be applied. This technique is applied to measure an error-tolerant, which is required [17].

  - Graph Edit Distance(GED): GED is one of the popular inexact matchings which computes the minimum required transformation

steps to edit G_1 to graph G_2. For comparing G_1 and G_2, GED tries to change both G_1 and G_2. The target graph G_2 will be created according to the erasing part of the G_1's nodes and edges. Moreover, GED will change the rest of the nodes' label and add the nodes and edges to G_2. Thus, after all, G_1 is altered to G_2.

2.1.4.3 Subgraph Matching

In finding the similarity between two graphs, there is another type of comparison named subgraph matching. If the comparison of the two graphs is limited to each subset, which describes each graph's most connected nodes, the subgraph matching arises.

There are some suggested techniques for subgraph matching that are listed below [40]:

- Approximate Constrained Subgraph Matching
- SAGA: Approximate Subgraph Matching using Indexing
- Mining Coherent Dense Subgraphs
- Tensor Analysis
- Subgraph Matching Via Convex Relaxation

Based on our proposed system, which is explained in Chapter 4, it is required to compute the similarity score between the presented family graphs and each

input sample to detect the sample's family. In Chapter 4 and Chapter 5, we will discuss our method for defining the similarity score in this research.

2.2 Concluding Remarks

This chapter provided an overview of the background knowledge, which is required for this thesis. In this chapter, we briefly review malware analysis, graph representation, fuzzy hashing, and methods of graph comparisons. After understanding the definitions mentioned above, in Chapter 3, we will review the related work done in the IoT malware world.

Chapter 3

Literature Review & Proposed Taxonomy

3.1 Overview

IoT security rapidly became a concern in both academic and industrial aspects due to IoT devices' growth. In this chapter, first, we review the different approaches in the IoT malware field to see which parts have been focused more on current studies. So, we group current studies in two different main categories based on the type of approaches. The following items display the discussing subjects in each section:

1. Non-Graph-Based Related Work
    (a) Network-Based Approaches

(b) Honeypot-Based Approaches

(c) Statistical-Based and String-Based Approaches

(d) Sensor-Based Approaches

(e) Energy-Based and Runtime-Based Approaches

2. Graph-Based Related Work

(a) Analysis Approaches

(b) Detection Approaches

Second, a comprehensive review of current security methods is essential for finding suitable solutions against IoT malware sabotage in real scenarios. After reviewing the related work, we express our proposed taxonomy of different defence mechanisms against IoT malware in this chapter.

3.2 Literature Review

In this section, we gather existing studies that are related to IoT malware detection or classification. This section will discuss Non-graph-based and graph-based approaches. Non-graph-based studies mostly focus on network-based studies and honeypot-based efforts. In this part, there are other types of non-graph-based works like statistical-based and string-based classification approaches, sensor-based studies, energy patterns, and runtime-based classification techniques. Afterward, we will examine graph-based studies under

two headings based on the approaches. So, first, we present IoT malware studies that are related to analysis. After that, IoT malware detection works will be discussed. In this section, we will review all the related studies that have been done in the IoT world to get a more accurate view of what has been done in IoT malware classification or detection. This literature review will help researchers find the existing gaps in the security of IoT. The number of machine-learning approaches in IoT malware detection is increasing, as are other types of studies. Nowadays, ML-based security schemes in IoT systems are so popular. Generally, malware detection methods are grouped into two main categories [66] including:

1. Behavior-based methods: Behavior-based approaches observe the behaviour of a binary file. Indeed, behaviour-based methods monitor the actual behaviour of the binary by running the sample. If the binary acts suspiciously on the system by monitoring some important predefined features like system calls, it can be known as malware. This technique focuses on how the binary performs after executing.

2. Signature-based methods: Signature-based approaches work based on matching the required code to the malicious patterns. Patterns or signatures typically are defined to describe a specific malware like binary features. This technique is popular for use in anti-malware systems.

Figure 3.1 shows some ML algorithms that can be used in malware detection approaches.

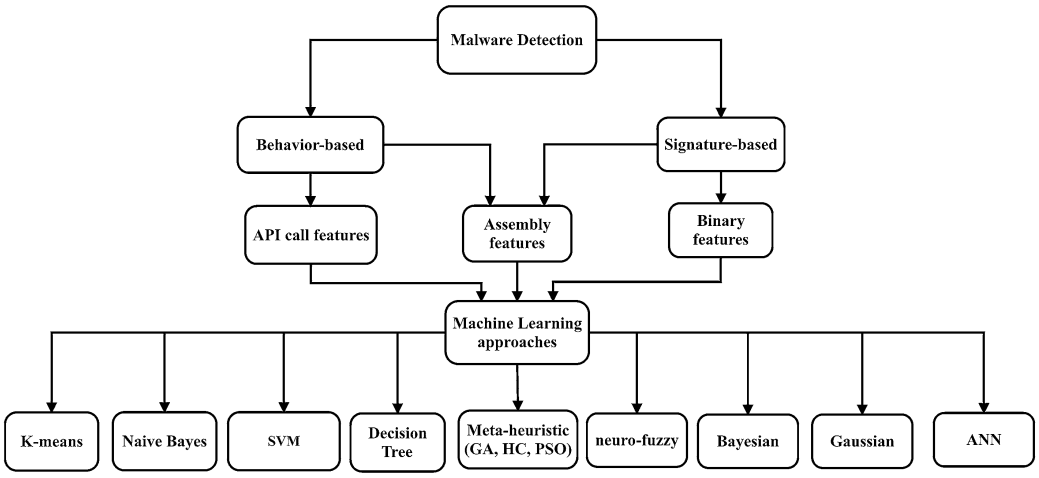


Figure 3.1: Different types of malware detection approaches [66]

In recent years, researchers have used machine learning-based approaches for IoT malware detection. Authors in [68] provides the below list to show which machine learning algorithm is effective in a specific case of IoT domain:

- Bayesian Theorem: It is statistics and has excellent implementations in the IoT domain. It is applied to estimate occupation of the battery running and of the room using PIR sensors [66].
- Genetic Algorithm: The Genetic Algorithm Placement of IoT Device (GA-PID) uses to find the place of IoT devices to perform the task.
- Reinforcement Learning: This technique learns from immediate interaction with the environment. So, this method is proper for IoT devices that have sensors due to making changes very fast.

- Ant Colony Algorithm: This technique is used in the IoT domain to find the path and make communication with other nodes due to the IoT features like irregular network topology, many nodes, the more variable network structure.

- Support Vector Machine Algorithm: In the IoT domain, supervised learning methods can be applied such as Support Vector Machines (SVM) for detecting a variety of attacks [73] and malware in the smart grid area [53].

- k-Nearest Neighbors (K-NN): The detection scheme in [15] uses K-NN to reduce energy consumption and save the maximum energy by 61.4% in comparison to the Centralized scheme. The overview of the detection scheme is shown in Figure 3.2.

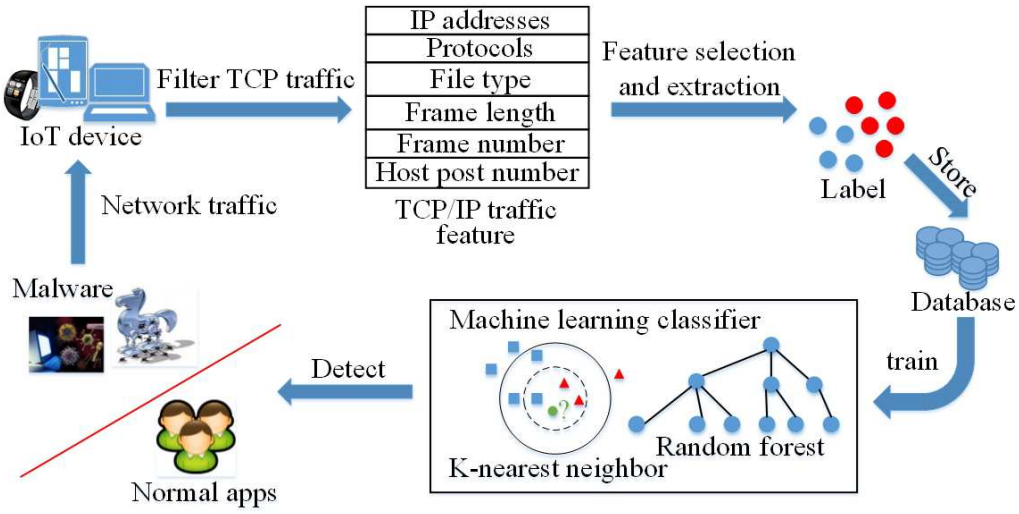


Figure 3.2: Illustration of the ML-based malware detection [72]

Further in [72], Narudin *et al.* (2016) uses K-NN and random forest as the classifiers. True positive (TP) for K-NN based malware detection is 99.7% and for random forest-based malware detection model is 99.9% [72].

3.2.1 Non-Graph-Based Related Work

This section provides an overview of non-graph-based approaches. Non-graph-based approaches mostly consist of several subsections, including network-based studies, honeypot-based strategies, statistical-based and string-based approaches, classification approaches, sensor-based studies, energy patterns, and runtime-based classification techniques.

3.2.1.1 Network-Based Approaches

Mirai is one of the prevalent IoT malware which operates according to build a large army for launching DDoS attacks on the victim devices. Mirai caused the most significant DDoS attack on October 21, 2016, and the target of Mirai is the Domain Name Service (DNS) server [41]. Other types of Mirai-like malware such as BASHLITE [21], Remaiten [71], and Hajime [29] use the same techniques as Mirai to compromise the victim systems, including the same brute force method, scanning IP addresses through the network to find an open port on the new target object, and trying to login by a dictionary of credentials [42].

Mirai and Mirai-like malware participate in several attacks like phishing and spamming [11] to gain access to the target or capture the required infor-

Table 3.1: IoT Malware DDoS Capabilities [24]

Malware	Year	Source Code	Agents CPU	DDoS architecture	DDoS attacks
Linux.Hydra	2008	Open Source	MIPS	IRC-based	SYN Flood, UDP Flood
Psyb0t	2009	Reverse Eng.	MIPS	IRC-based	SYN Flood, UDP Flood, ICMP Flood
Chuck Norris	2010	Reverse Eng.	MIPS	IRC-based	SYN Flood, UDP Flood, ACK Flood
Tsunami, Kaiten	2010	Reverse Eng.	MIPS	IRC-based	SYN Flood, UDP Flood, ACK-PUSH Flood, HTTP Layer 7 Flood, TCP XMAS
Aidra, LightAidra, Zendran	2012	Open Source	MIPS, MIPSEL, ARM, PPC, SuperH	IRC-based	SYN Flood, ACK Flood
Spike, Dofloo, MrBlack, Wrkatk, Sotdas, AES.DDoS	2014	Reverse Eng.	MIPS, ARM	Agent-Handler	SYN Flood, UDP Flood, ICMP Flood, DNS Query Flood, HTTP Layer 7 Flood
BASHLITE, Lizkebab, Torlus, Gafgyt	2014	Open Source	MIPS, MIPSEL, ARM, PPC, SuperH, SPARC	Agent-Handler	SYN Flood, UDP Flood, ACK Flood
Elknot, BillGates	2015	Reverse Eng.	MIPS, ARM	Agent-Handler	SYN Flood, UDP Flood, ICMP Flood, DNS Query Flood, DNS Amplification, HTTP Layer 7 Flood, other TCP Floods
XOR.DDoS	2015	Reverse Eng.	MIPS, ARM, PPC, SuperH	Agent-Handler	SYN Flood, ACK Flood, DNS Query Flood, DNS Amplification, Other TCP Floods
LUABOT	2016	Reverse Eng.	ARM	Agent-Handler	HTTP Layer 7 Flood
Remaiten, KTN-RM	2016	Reverse Eng.	ARM, MIPS, PPC, SuperH	IRC-based	SYN Flood, UDP Flood, ACK Flood, HTTP Layer 7 Flood
NewAidra, Linux.IRCTelnet	2016	Reverse Eng.	MIPS, ARM, PPC	IRC-based	SYN Flood, ACK Flood, ACK-PUSH Flood, TCP XMAS, Other TCP Floods
Mirai	2016	Open Source	MIPS, MIPSEL, ARM, PPC, SuperH, SPARC	Agent-Handler	SYN Flood, UDP Flood, ACK Flood, VSE Query Flood, DNS Water Torture, GRE IP Flood, GRE ETH Flood, HTTP Layer 7 Flood

mation at the first step of planning a specific major attack. [24] provides a comparison and analysis related to the recent IoT malware families. Further, Different types of DDoS attacks are discussed in [24]. Table 3.1 displays pieces of information about DDos-related malware families such as CPU types and architecture.

Reverse engineering techniques have found the most available source codes of IoT malware samples. Authors in [24] discuss the correlations between

the malware families. De *et al.* (2017) mention "Linux.Hydra" is the first DDoS-related IoT malware. And afterward, the source code of Linux.Hydra has evolved into three other malware within the years. For example, Tsunami is a version of Linux.Hydra that has been changed to create other malware samples named "chunks of Remaiten" and even "NewAidra". Figure 3.3 shows the relationships between the malware families.

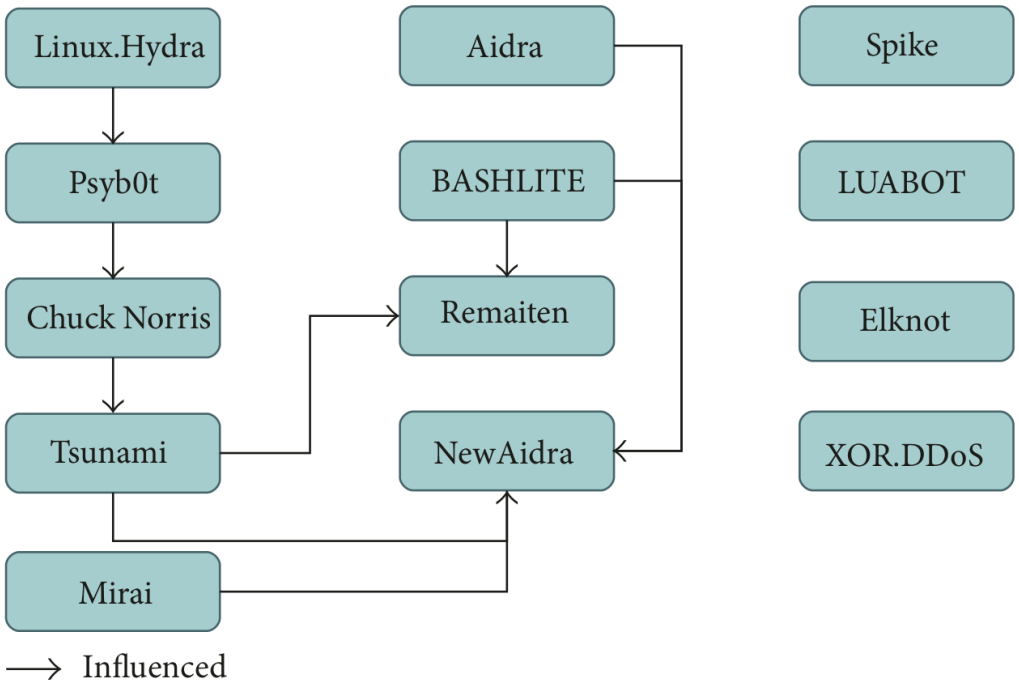


Figure 3.3: IoT DDoS Capable Malwares - Correlations [24]

In a similar work [28], Gopal *et al.* (2018) analyze Mirai malware. They provide a white-list to prevent IoT botnets from being expanded by scanning all router applications to catch each application's hashes and store them

in the database. If the hash does not match, the application gets blocked. Likewise, [37] discusses Mirai and other similar worms in a comprehensive review of the IoT botnet domain.

In a recently published paper [42], a network-based algorithm is designed for detecting IoT Mirai bots or similar malware's bots in large-scale networks. Kumar *et al.* (2019) provide emulated infected IoT devices by Mirai in their testbed. After that, a total of transmitted packets, about 1.5k packets got captured for further analysis. Analyzing all the packets shows:

- All packets are TCP SYN packets.
- Destination of the majority of the scanning packets, 90% of all, is port 23, and others (10%) is port 2323.
- PSH+ACK is a keep-alive message between a bot and C&C server, which is a kind of Mirai signature.

Authors in [42] analyze Mirai malware characteristics and use testbed measurements and simulations to study the relationship between bot detection delays and the sampling frequencies for device packets.

This research [69] models two IoT malware named Hajime and Mirai that infect IoT devices. Furthermore, Tanaka *et al.* (2017) use two different network topologies to show the effect of topology on the rate of infection by Mirai.

Chang *et al.* (2017) [20] propose an IoT sandbox which supports nine different CPU architectures such as ARM, MIPS, MIPSEL, PowerPC, Sparc x86,

x64, and sh4. This IoT sandbox is used for analyzing malware behaviour and consists of five elements and four processes, as shown in Figure 3.4, which is implemented in Vmware (Ubuntu 14.04).

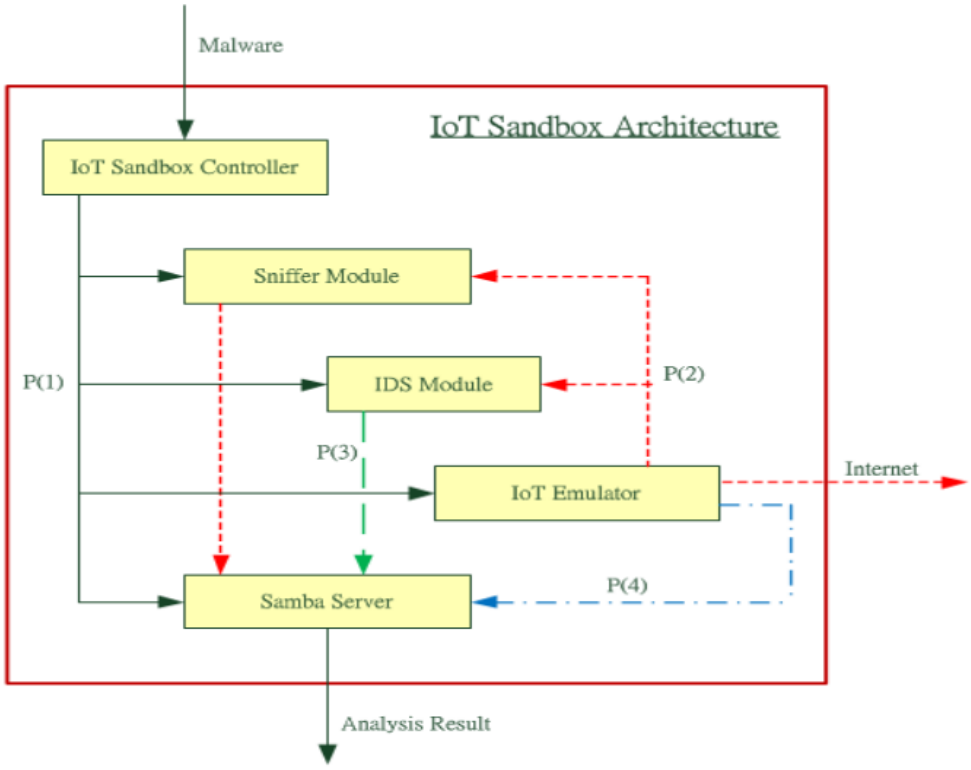


Figure 3.4: IoTBox Architecture [20]

Five elements in the IoT sandbox are as follows:

1. IoT Sandbox Controller: It is responsible for starting up the modules, duplicate malware to the Samba server, and shut the module down after every five minutes.

2. Sniffer module and IDS module: It is responsible for recording and Analyzing the packets. Two types of Snort are used here. One of them is on sniffer mode, and the next one is IDS (Intrusion Detection System).

3. IoT Emulator: It is based on Qemu. It is a virtual machine that can support several CPU architectures and many types of OSs.

4. Samba server: Samba server's task is related to connecting IoT emulator and modules.

Four processes of the IoT sandbox are as follows:

1. Initialize Process (P1): IoT sandbox controller has been discussed before. It starts a module and copies malware to the Samba server. So, IoT Emulator has access to malware from the Samba server. Therefore, it can analyze and save the malware in the Samba server.

2. Network Base Analysis Process (P2): Due to monitoring the connection of malware and the C&C server, the Sniffer module sniffs packets and records the behaviour of the network.

3. Exploit and Spread Analysis Process (P3): For generating IDS rules, the IDS module uses several signatures for spread method detection and exploitation.

4. Host Base Analysis Process (P4): IoT emulator has access to the malware on the Samba server and uses the Strace command to run it. Afterward, this process saves the behaviour of the malware on the Samba server. Strace can keep a trace of system calls and record the malware's behaviour, including which files have been modified, read, and written by malware.

There is not a very high number of machine learning-based approaches in the network aspect of IoT devices. IoT traffic is not like other Internet-connected devices because IoT devices mostly communicate with particular network components instead of connecting to different web servers like other devices. IoT devices' network traffic pattern is repetitive due to routine pings with small packets at a specific time for logging.
[64] provides a method based on monitoring network traffic at flow-level granularity. Sivanathan *et al.* (2016) suggest that detecting threats with flow-based features is very efficient because of decreasing the cost of investigating packets at a deep-level. Characteristics of the framework for a smart home gateway router are as follow:

- Lightweight Features: Due to handling high bandwidth traffic by routers, all generated features should be lightweight [31]. The algorithm in high scale bandwidth should be based on network flow statistics.

- Protocol Agnostic Features: Routers should precede different types of packets (e.g. TCP, UDP, HTTP). Thus, the algorithm should work to

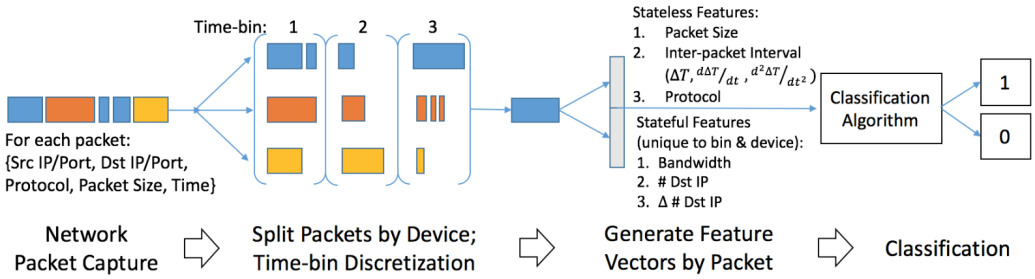


Figure 3.5: IoT DDoS detection pipeline [25]

rely on features that are common in all types.

- Low Memory Implementation: Because of memory limitations, the algorithm should be optimal. So, the algorithm is stateless or needs to store flow information over short time windows [64].

The purpose of this paper [25] is identifying the traffic of DoS (Denial of Service) attacks in smart home LAN. In doing so, Doshi *et al.* (2018) implement a machine learning framework for network traffic of IoT devices. This framework has the following four phases, which are shown in Figure 3.5.

- Traffic Capture: This part saves source IP address, source port, destination IP address, destination port, packet size, and timestamp of all packets from smart home (IoT) devices.

- Grouping of Packets by Device and Time: After saving all IoT packets, packets will be grouped based on the source IP address. Each device's packets are divided into non-overlapping time windows according to the timestamps.

- Feature Extraction: Features are generated according to the IoT device behaviour in the network. The features have two types here:
  - Stateless Features: packet size, inter-packet interval, protocol.
    1. Packet size: Benign packet size is between 100 B and 1200 B. Otherwise, the size of the majority of malicious packets is under 100 B.
    2. Inter-packet interval: About IoT devices, typically, the time between sending packets is at regular intervals. Because IoT devices mostly send network pings or other automated network activities.
    3. Protocol: Distribution of different types of protocols is different between regular traffic and malicious traffic. UDP packets are three times more than TCP packets in a normal situation because of UDP video streaming. In malicious network traffic, the ratios of packets of two protocols are the same.
  - Stateful Features: bandwidth, number of different destination IP addresses within a 10-second window, and calculate the change in the number of different destination IP addresses between time windows.
    1. Bandwidth: Usage of bandwidth can show the behaviour of IoT devices in the network. This method separates the network traffic based on the source device and computes band-

width within 10-second time windows to determine the immediate bandwidth related to each device.

2. Count of different destination IP addresses within a 10-second window: IoT devices usually communicate with a few endpoints. Thus, increasing the number of endpoints can be a sign of an attack in the traffic.
3. Calculate the change in the number of different destination IP addresses between time windows: This may reveal an entirely new endpoint that possibly indicates an attack.

- Binary Classification: Implementing the machine learning techniques including K-nearest neighbours, random forests, decision trees, support vector machines using the Scikit-learn Python library, and the Keras library for deep neural networks have been used.

3.2.1.2 Honeypot-Based Approaches

In recent years, several research efforts have used a honeypot approach in the IoT malware domain. Certainly, the honeypot-based effort is a popular method to discover zero-day attacks. Most honeypot approaches for IoT devices are low-interaction [44], which means the honeypots provide a limited interaction. So, this kind of honeypot can receive limited information. Hence, Luo *et al.* (2017) [44] build a high-interaction honeypot. For building a high-interaction honeypot, there are two possible scenarios:

- Physically: It is not a practical way because of various types of IoT devices.
- Virtually: Lack of emulators for IoT devices makes it impossible.

To overcome the challenge, Luo *et al.* (2017) propose a generic framework based on an intelligent-interaction honeypot. This proposed honeypot can answer the requests depending on the learning process. Figure 3.6 represents the proposed system, which consists of four components, and each part operates separately to make the system intelligent due to the learning phase.

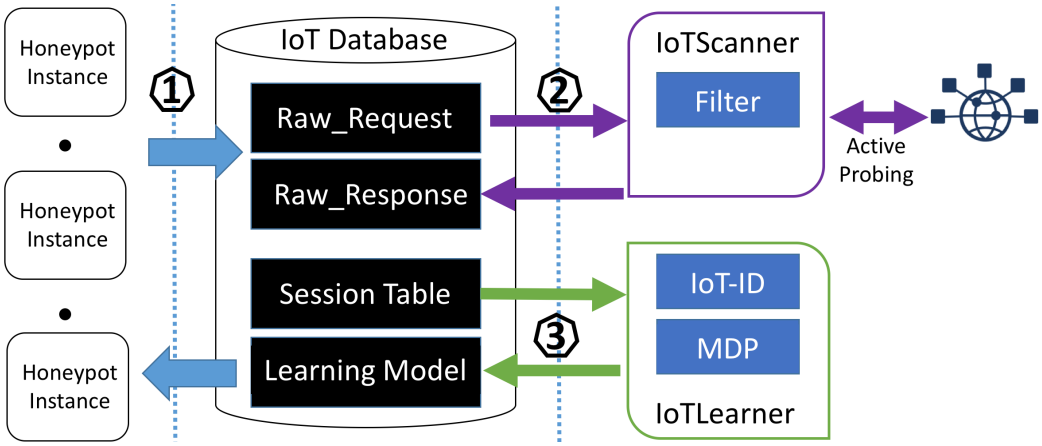


Figure 3.6: Overview of the system in [44]

1. IoT-Oracle: Store any data which is related to IoT devices.
2. Honeypot Module: It is deployed to Amazon AWS and Digital Ocean. This part is responsible for capturing the network traffic of attack and

keeping the communication alive with the adversaries to interact. All requests and answers are saved on the IoT-Oracle component.

3. IoTScanner: It is responsible for searching the Internet to find any IoT devices that can give an appropriate answer to the request. All obtained information is stored in the database as well.

4. IoTLearner: Machine Learning (ML) techniques are applied to train a model to rely on the attacker's interaction. After some rounds of the learning process, the system can provide an optimized model to give the attacker an appropriate answer.

For each query, IoTScanner searches the Internet to find a rational answer. So, IoTScanner can find at least hundreds or even thousands of possible responses, and all answers get archived in the candidate pool.

Thus, IoTLearner first selects several answers from that pool randomly, monitors the attacker's reaction after receiving the response, and saves it in the database for further analysis. If the attacker keeps the interaction, it means the answer is satisfying, and the attacker believes that it is a real vulnerable device to send malicious files to it. Since all interactions are stored, machine learning techniques can select a correct response from the dataset.

In the first step, this system operates like a low-interaction honeypot because the system is not familiar with IoT devices and malware behaviours. Interaction with attackers is not a complicated process at all. For example, most attackers try to do some actions like HTTP requests, injecting commands

without authentication, and capturing the scripts or malicious codes. All these tasks are not too difficult to catch.

In another work [27], authors use an open-source honeypot named "Cowrie SSH/Telnet Honeypot" [52] to take the unique information from malicious IoT traffic behaviours for detecting infected devices by Mirai botnet.

Telnet-based attacks for targeting IoT devices have increased since 2014 [56]. An IoT honeypot and sandbox are presented in [56] which can attract telnet-based attacks related to IoT devices on the variety of CPU architectures such as ARM and MIPS. Pa *et al.* (2016) discusses five types of DDoS malware families on telnet-enabled IoT devices. The malware samples support a variety of architectures. The total number of malware samples in this research is 106. 88 out of 106 samples are not in the VirusTotal [4] database, and they are completely new. Besides, none of the 57 antivirus software of the VirusTotal can recognize 2 out of each 18 samples in the VirusTotal. The proposed IoTBOX in [56] supports eight different CPU architectures including MIPS, MIPSEL, PPC, SPARC, ARM, MIPS64, sh4, and X86.

3.2.1.3 Statistical-Based & String-Based Approaches

The authors in [10] use statistical and string features that can be extracted easily to develop IoT malware signatures. Alhanahnah *et al.* (2018) use the features to cluster IoT malware families and distinguish between malicious and benign samples.

This research works on two datasets which are collected by the IoTPOT team

[54], and each sample has both time label and MD5 name:

- First dataset: It contains 1150 malware samples of three months (May 2016-August 2016).
- Second dataset: It contains 4000 malware samples of one year (October 2016-October 2017).

Alhanahnah *et al.* (2018) work on 5150 malware samples. The existing number of IoT malware is about 7000, and this research covers about 74% of the total is known IoT malware. So, the result of this research is reliable due to reflecting the real characteristics of IoT malware.

Jaccard similarity (on statistical features) calculates the inter-cluster similarity score among all clusters. Moreover, the reverse of Euclidean distance is being used for finding the similarity among the string features. After calculating the Overall Similarity (OS), the clusters with an OS score higher than the threshold can be merged. Alhanahnah *et al.* (2018) have found the threshold in experimental analysis.

Working on real IoT devices and dynamically analyzing IoT malware are proposed for their future work.

Figure 3.7 shows the proposed method consists of two sections:

- Offline part which is generating signatures.
- Online part which is detecting /classifying malicious and benign files.

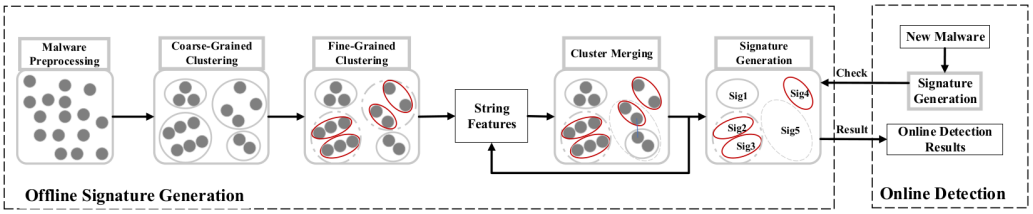


Figure 3.7: System overview in [10]

3.2.1.4 Sensor-Based Approaches

Sikder *et al.* (2018) in [62] discuss sensor-based threats that IoT devices possibly face. Compromising the sensors is one of the ways that adversaries can attack an IoT system. For doing so, transferring malicious code or even triggering the commands to active malicious code is required to target the IoT sensors. For instance, attacking through voice assistant applications like Apple's Siri and Google voice search compromises microphones by Cyber-Physical Voice privacy Theft (CPVT) Trojan horse. CPVT is a kind of malware that can be used for recording the message. Attackers use command-lines to find the appropriate time for recording. So, attackers reach the target device for accessing the recorded through one of the communication ways, including WiFi, Bluetooth, and sensory channels.

The other way to send malicious commands is by controlling the voltage of a light source. Changing the voltage range can be performed as a trigger for IoT malware in some devices for getting activated or changing the magnetic field. In [32], Hasan *et al.* (2013) demonstrates that changing the light intensity of the TV screen or laptop monitor can trigger a message to compromise an

IoT device. Consequently, the results show that audio sensors can transfer an embedded message in an audio song to compromise a smartphone.

3.2.1.5 Energy-Based & Runtime-Based Approaches

Machine learning approaches in the IoT malware hunting domain mainly focus on extracting features from the opcodes [12] and the pattern of energy consumption [13]. The technique presented in this research [61] is a malware classification approach based on the frequency of opcodes to classify malware families. In [59], the author presents a similarity graph to detect metamorphic malware that depends on the application's opcodes. Azmoodeh *et al.* (2018) [14] provide a detection mechanism for IoT crypto-ransomware in the networks. Azmoodeh *et al.* (2018) propose a machine learning-based technique, which is an example of using energy consumption footprint in the IoT malware hunting domain. The authors monitor all existing processes' energy usage patterns to distinguish between ransomware and benign software in Android devices in IoT networks. So, a different range of energy usages in a victim device can be used as a feature in IoT malware detection [19].

Additionally, authors in [58] present a method for malware classification. In this technique, all samples are placed in a group according to the runtime behaviours. Pekta *et al.* (2017) monitor some behaviour-based features during the execution, including the file system, network, registry activities, and API-call sequences. This framework has three main steps, as shown in Figure 3.8.

1. Behavior of samples is extracted by running in two sandboxed environments named VirMon [70] and Cuckoo. Moreover, the Application Programming Interface (API) calls and Operating System (OS) changes can reflect a file's behaviour.

2. All of the extracted features are applied to the analysis report.

3. The malware dataset is divided into training and testing groups.

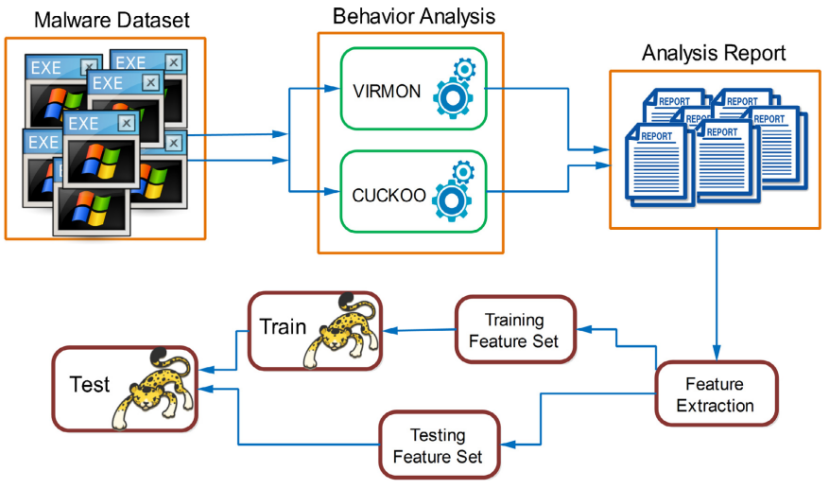


Figure 3.8: Overview of the proposed malware classification system [58]

3.2.2 Graph-Based Related Work

This part examines the existing studies that work on graph-based solutions in the IoT malware world. We break the current works into the following two subsections based on proposing analysis results or providing detection models.

3.2.2.1 Analysis Approaches

Authors in [7] analyze Android and IoT malware samples to define the similarities and differences based on the graph-based method. They use IoTPOT dataset [55] including 2874 IoT malware samples and 201 Android malware samples for comparing Android malware and IoT malware. Alasmary *et al.* (2018) provide an in-depth analysis of malware graphs and extract CFGs for both groups of malware and analyze the samples based on graph-based features including the number of nodes and edges, average shortest path, betweenness, closeness, and density. The authors suggest using machine learning techniques according to graph-based features from FCGs to classify the samples in their future work. Moreover, Alasmary *et al.* (2018) use the IoTPOT dataset [54] which has 201 Android malware samples and 2874 IoT Malware samples.

Alasmary *et al.* (2019) [8] uses the CyberIOC dataset [2] including 2891 Android malware and 2962 IoT malware in their research. The approach of [8] is based on generating CFGs and extracting graph-based features from the CFGs. After generating the graphs, Alasmary *et al.* (2019) compare IoT graphs and Android graphs to define their similarities and differences. For instance, the results show density, closeness, betweenness, and the number of nodes is high in Android samples compared to IoT samples. However, the number of edges and complexity are higher in IoT than in Android malware samples. So, graph-related features can represent the software very well for other classification techniques.

Besides, [9] uses IoTPOT dataset [55] including 2347 IoT malware samples and 261 benign IoT software and 2891 Android malware samples in their research. Alasmary *et al.* apply different ML techniques such as LR, SVM, RF, and CNN deep learning methods on the dataset to classify IoT malware and IoT benign software. Thereafter, Alasmary *et al.* compare the CFGs for IoT and Android malware to discuss the similarity and differences by using graph-based features.

Features that Nguyen *et al.* (2019) use in [48] are called Printable String Information (PSI). PSI features are generated from PSI-graphs, which shows the path of binary execution. Then, working on the PSI-graph instead of the whole graph is a new approach in graph-based studies due to reducing the system's cost and complexity compared to other previous works [48]. On the other hand, small graphs (PSI-graphs) can be analyzed instead of huge CFGs. After extracting the PSI-graphs, Nguyen *et al.* (2019) do the analysis and classification on the PSI-graphs instead of the complete FCGs. Moreover, Nguyen *et al.* (2019) work on 7199 IoT botnet samples and 4001 benign software captured from the IoTPOT [54]. Furthermore, Nguyen *et al.* (2020) in [50] introduce an IoT botnet detection system that supports two types of architecture for IoT, including ARM and MIPS. They apply five machine learning algorithms, including SVM, RF, DT, and kNN, on about 10k samples, containing 6165 IoT botnet samples and 3845 IoT benign samples.

3.2.2.2 Detection Approaches

The authors in [67] work on the CNN technique for classification, representing a lightweight technique for DDoS malware detection in IoT. To do this, Su *et al.* (2018) convert the samples to images and then extract features from the generated images. For example, a one-channel gray-scale image representing a malware's binary can be used for feature extraction. Su *et al.* (2018) use 500 malware samples of the IoTPOT [54] dataset. This approach has a high accuracy of 94% for classifying malicious and benign IoT samples. Accordingly, the proposed classification in [67] can group malware samples in two main families (Gafgyt and Mirai) with an accuracy of 81.8%. The authors list the different techniques for image extraction in the paper. Also, adding static features like opcodes sequences and API calls can be done in future work for better results.

Thereafter, in [47], authors gather 9342 IoT malware samples to create a dataset for further analysis. In this approach, malware samples convert to gray-scale images to extract features to perform malware family classification. So, they design a malware classification based on binary image representation.

This paper [35] works on measuring the similarity between IoT malware samples. The approach in [35] is disassembly-based for the classification of Linux-based and IoT malware samples. The authors work on 8,713 IoT malware from the IoTPOT [54] dataset. For the disassembly-code based similarity, sequences of an opcode for each function are extracted as a feature

of Linux-IoT malware. Moreover, this approach uses "t-SNE" to visualize IoT malware samples on a two-dimensional plane. In the future study, Isawa *et al.* (2018) plan to make a classifier that takes disassembly code for Linux-based and IoT malware as an input of the system. Isawa *et al.* (2018) use a significant number of IoT malware samples so that the results can reflect the real IoT malware's real behaviour. The results represent that just 1.76% of 8713 samples are packed Isawa *et al.* (2018). The authors confirm that the majority of IoT malicious software samples are not packed. On the other hand, static analysis can be useful and beneficial for just Linux-IoT malware samples due to referring to the result [35].

HaddadPajouh *et al.* (2018) [30] built a dataset in this study which consists of 280 IoT malware and 271 benign samples. The proposed system in [30] is based on deep recurrent neural networks for detecting IoT malware by using opcodes sequences and patterns of energy consumption as the features on the Information Gain (IG) feature selector. HaddadPajouh *et al.* (2018) suggest the using the current system in a real environment and analyzing the real behaviour of new malware samples as the future work. Their proposed method is shown in Figure 3.9:

Azmoodeh *et al.* (2018) [13] collect 1078 benign samples and 128 IoT malware to create a dataset to make their dataset in this research. The approach in [13] is based on deep learning by using the n-gram of opcodes as the list of features to detect IoT malware by IG. they select the features by

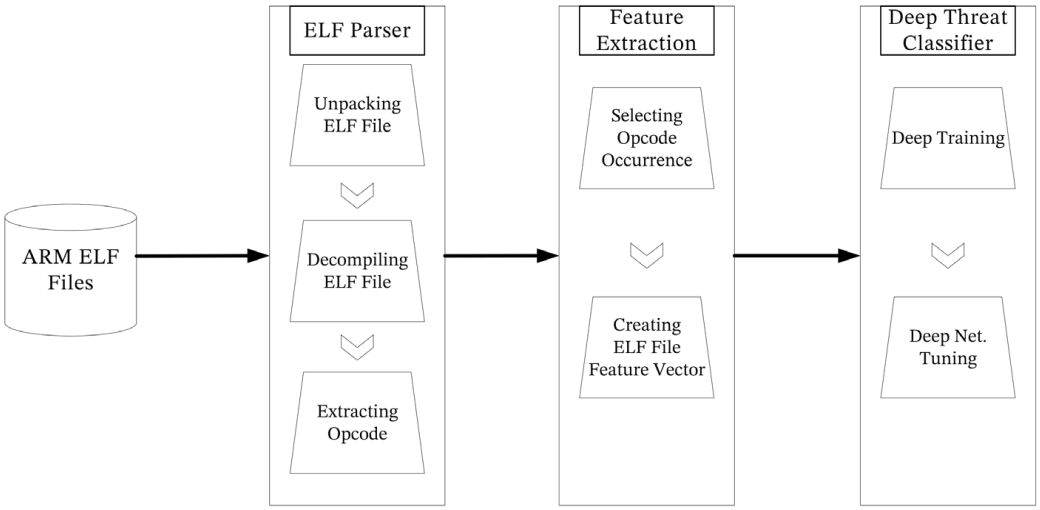


Figure 3.9: Proposed deep IoT threat hunting approach [30]

ranking them according to the amount of available information contained in a classification problem. Using a real-world IoT which consists of a large number of diverse data is the next research goal.

In [57], graph-based access control is created, which can be run as a module on IoT nodes or in the network. Pahl *et al.* (2018) use a group of graph-based features like node counts, edge counts, density, degree centrality, betweenness centrality, closeness, diameter, and distribution of the shortest path.

In [5], authors use CNN classifier to detect IoT botnet based on the features from the PSI-graph. Abusnaina *et al.* (2019) work on 2281 IoT malware samples from the CyberIOC dataset. [2]. So, instead of analyzing the whole graph, Abusnaina *et al.* (2019) just work on the specific parts of the graphs that include useful information.

Likewise, Nguyen *et al.* (2018) [49] propose an approach based on PSI-

graphs to collect information for static analysis. This proposed system uses a graph convolution neural network classifier for the detection of IoT malware. Nguyen *et al.* (2018) use 4002 IoT malware samples from the IoTPOT dataset [54] in this research. Their system works based on generating PSI-graph to consider the linkages between PSI, which is essential for static analysis.

The authors in [23] use the opcodes sequence to propose a detection system for IoT malware. Darabian *et al.* (2020) apply some ML techniques including k-Nearest Neighbors (KNN), SVM, random forest, decision tree, and AdaBoost on a dataset (247 IoT malware and 269 Benign samples) from [30]. Maximal Frequent Patterns (MFP) of opcodes can distinguish between IoT malware and benign samples effectively. So, Darabian *et al.* (2020) use the most frequent opcodes sequences that have been detected by sequential pattern mining techniques to classify the IoT malware, benign IoT software, and polymorphic malware.

[51] combines ML and deep learning to generate novel features based on PSI-rooted and sub-graph features for cross-architecture IoT botnet malware detection in a completely static way. Nguyen *et al.* (2019) use 6164 IoT malware and 3779 non-malicious samples from IoTPOT [55] and VirusTotal [4] in their study. They achieve an accuracy of about 97% for their proposed PSI-graphs-based detection system. Moreover, Dynamic analysis is mentioned as future work.

3.3 Feature-based IoT Malware Taxonomy

In the following parts of this section, first, we will explain the motivation that encourages us to present this study. After that, we will propose the novel taxonomy that is the first attempt to classify the existing related research efforts in the IoT landscape to the best of our knowledge. Furthermore, we will study the different features that the most related previous works use to detect or classify malicious samples.

3.3.1 Motivation

Compared to the dramatic growth of IoT malware attacks, related studies and proposing helpful techniques have grown less rapidly in the IoT field. However, there are several excellent pieces of works that present the solutions for classifying malicious IoT software and non-malicious IoT software. Besides, part of the previous studies focuses on comparing IoT malware and Android malware based on the extracted features. Each of the earlier works follows a different approach to see the required results, so all those techniques can be categorized into particular groups based on their approaches. Thus, we propose a novel feature-based taxonomy that covers different IoT malware-related studies' techniques.

To the best of our knowledge, it is a novel overview of IoT malware detection methods. This proposed taxonomy can help researchers to get a big picture of existing attempts to understand the advantages and disadvantages of security

methods in IoT for finding the research gaps. Furthermore, we study the features that have been used in IoT malware detection or analysis approaches. This study gives the other researchers a better understanding of common prevention and detection methods against IoT malware attacks to see which parts require more research efforts.

3.3.2 Taxonomy Overview

Figure 3.10 shows the proposed taxonomy which classifies different IoT malware detection approaches in different subgroups. In this taxonomy, we put some sub-categories of each main category that we find in the literature. We provide several examples for each sub-category in Figure 3.10.

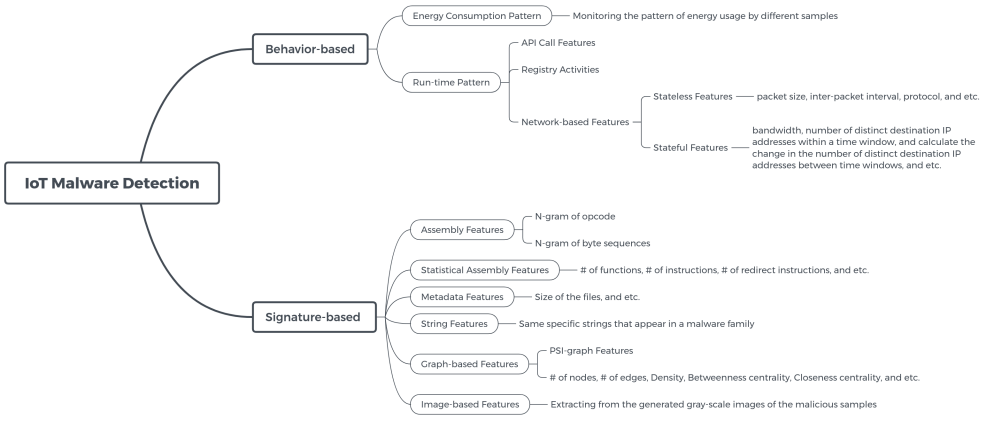


Figure 3.10: Proposed Feature-based IoT Malware Taxonomy

Based on the study of the related work that we completely discussed in Chapter 3, we can break all proposed IoT malware detection approaches into two following main categories:

1. Behavior-based approaches: In this dynamic analysis, the researchers attempt to activate the malware samples in an isolated environment like a sandbox for understanding the actual behaviour or potential behaviour of that malicious object. Any abnormal and unusual behaviour or unauthorized actions are marked as malicious or suspicious during the analysis. There are two sorts of methods in this group:

    - The first method is about the **energy consumption patterns**, which monitors the energy consumption patterns of different processes to distinguish between malicious and benign software. For example, authors in [12] and [14] use energy consumption patterns as a feature in their study.
    - The second method is based on finding the **run-time patterns**, which mostly uses network features, registry activities, and API call features. For example, [64] and [25] use network-based features in their research. Moreover, a combination of network-based features, registry activities, and API calls are the group of behaviour-based features in [58].

2. Signature-based approaches: Getting familiar with the specific attributes of a sample is known as the signature-based approach. This approach helps bring up the specific digital fingerprints of the executed binaries, which can be called the sample's signature. So, it is a fast method to determine the signature for known malware. Also, it can be used in

many security tools to prevent known attacks. This group can apply different types of features such as n-gram of opcodes, string features, statistical features, and graph-based features in their approach.

- Examples of using n-gram of opcode: [13], [35], and [23]
- Examples of using PSI-graph: [48],[49] [5], and [51]
- Examples of using image-based features: [67] and [47]
- Examples of using graph-based features: [7], [8], [9], and [57]

Furthermore, in [30], the authors use a combination of behaviour-based and signature-based features. They use n-gram of opcode and energy consumption patterns as their features.

The number of studies that work on behavioural methods purely is few in comparison to signature-based techniques. The main reason is that IoT malware supports different types of CPU architecture, so providing a simulation environment in the IoT domain is much harder than non-IoT malware. So this characteristic of IoT malware may cause the problem in behaviour-based approaches. Triggering the IoT malware samples to show the real malicious activities is another problem that makes the behavioural attempts time consuming and inaccurate. Besides, most of the IoT malware samples are botnet and infect a bunch of connected devices. The adversaries take complete control of the victim devices by the use of the IoT zombies army. Transferring the required commands via C&C communication helps the attackers to create massive sabotage in the future like a major DDoS attack. Since most of

the C&C servers of known IoT malware samples are down, it makes another obstacle against IoT behaviour-based methods.

Besides, reviewing prior works expresses that although graph-based features are effective for classifying between "IoT malware Vs. Android malware" or "Benign Vs. Malicious software", they are not proper for IoT malware families classification. The reason might refer to the differences between benign and malicious software's structures that graph representation can distinguish them well. Also, the structure of IoT and Android malware is different in comparison to the different IoT malware families. So, graph representation cannot determine the family of IoT malware for just a small part of changes that may happen in their codes and lots of similarities that may contain.

As our second (Con_2) contribution of this research, we introduced a novel feature-based taxonomy of IoT malware detection in this section. Also, we gather different groups of features in previous studies as our third (Con_3) contribution.

3.4 Concluding Remarks

In this section, we summarize the recent IoT malware studies based on their approaches. Table 3.2 displays graph-based and non-graph-based similar studies that have the most similarity to our research.

Extracting features from the PSI-graph helps to analyze the essential parts

Table 3.2: Summery of Previous Related Work in IoT Malware Area

Ref.	Approach	Graph	Fuzzy Hash
[7]	Compare android and IoT malware	FCG	No
[48]	IoT malware detection on the PSI-graphs instead of the original FCGs	PSI	No
[67]	Malware samples convert to gray-scale images to extract features for malware family classification	No	No
[35]	Measuring the similarity between IoT malware samples by sequences of an opcode for each function	No	No
[30]	Detection of IoT malware by using opcodes sequences and patterns of energy consumption	No	No
[13]	IoT malware detection by using the n-gram of opcodes	No	No
[57]	Creating graph-based access control by using graph-based features	FCG	No
[10]	Using string features for distinguishing between benign and malicious IoT software	No	No
[5]	IoT botnet detection by PSI-graphs	PSI	No
[49]	Static analysis on IoT samples	PSI	No
[23]	Detection system for IoT malware based on OP-Code	No	No
[51]	Combination of ML and deep learning based on PSI-graph	PSI	No

of binary codes, which are the keys of detection, not the entire code. So, PSI-graph might not be accurate enough, depending on the types of approaches. Mostly extracting features from either PSI-graph or n-grams of opcodes is related to static analysis of the malware source code, not malware's behavioural characteristics. Graph-based features have not been applied as much as other types of features for the IoT malware detection/classification. Several studies use FCGs for comparing the IoT malware and Android malware and finding similarities between them. None of the previous studies works on

the family classification on IoT malware, and also no work takes advantage of fuzzy hashing techniques for finding similarities in IoT malware samples. This thesis proposes a framework based on a combination of fuzzy hashing and CFGs to classify IoT malware samples according to the family structure. This thesis addresses some important gaps in previous similar works in the IoT malware landscape. For example, our proposed framework classifies a variety of malware families based on the family structure. Consequently, we introduce a weighted merged hash graph for each IoT malware family that reflects each family's structure. To the best of our knowledge, it is a novel way to work on identifying IoT malware families by using the combination of call graphs and fuzzy hashing. In Chapter 4 and Chapter 5, we will examine specifically our proposed framework and details of implementation.

Furthermore, in this chapter, we provide a novel taxonomy of different IoT malware detection approaches to the best of our knowledge (Con_2). Second, we study the existing literature's various features (Con_3). Indeed, we study the features in IoT malware works according to our proposed taxonomy.

To the best of our knowledge, this study is the first endeavour to categorize the existing research efforts on IoT malware through a taxonomy that we hope other researchers will extend comprehensively in the future. The following chapter, Chapter 5, represents the details of the implementation of our proposed framework.

Chapter 4

Proposed Framework

4.1 Overview

In this chapter, a novel graph-based IoT malware classifier is proposed to group IoT malware samples into the appointed families. The proposed framework can be used as an automated IoT malware classification tool to classify a new IoT malware sample. Indeed, it can help other researchers to study different IoT malware families to provide new appropriate solutions in real security scenarios.

Moreover, this framework represents an Aggregated Weighted Graph of Hashes (AWGH) for the first time for IoT malware families. AWGH is a merged graph of all samples' hash graphs in a family that reflects each IoT malware family structure. AWGH can help security analysts save critical time by detecting a new malware sample's assigned family. In the following sections,

we will explain our motivation for the proposed method. After that, we will discuss the overview of the proposed framework and explain all phases in detail.

4.2 Motivation

Previous studies show that malware detection is an important issue, especially for IoT. Because the number of IoT devices is increasing rapidly and IoT devices are becoming new targets for adversaries. Most of the previous studies have focused on extracting general graph-based features from CFGs or just using a combination of statistical and string features in their proposed detection mechanisms. Furthermore, previous studies have worked on distinguishing between malicious IoT software and benign IoT software in most cases.

Therefore, at the time of proposing this idea, classifying the malware samples into their belong families is a new challenge in IoT, especially when the method is graph-based. Moreover, similarities and differences of various malware types (like Android malware and IoT malware) have been analyzed based on the statistical features in several previous papers. Although there are several graph-based studies available, none of them considers graph-based family classification. Our proposed framework addresses this problem by detecting malware families based on the introduced AWGH that represents the IoT malware family structure.

4.3 Framework Overview

Figure 4.1 outlines the general proposed framework. This framework consists of three main phases, as follows:

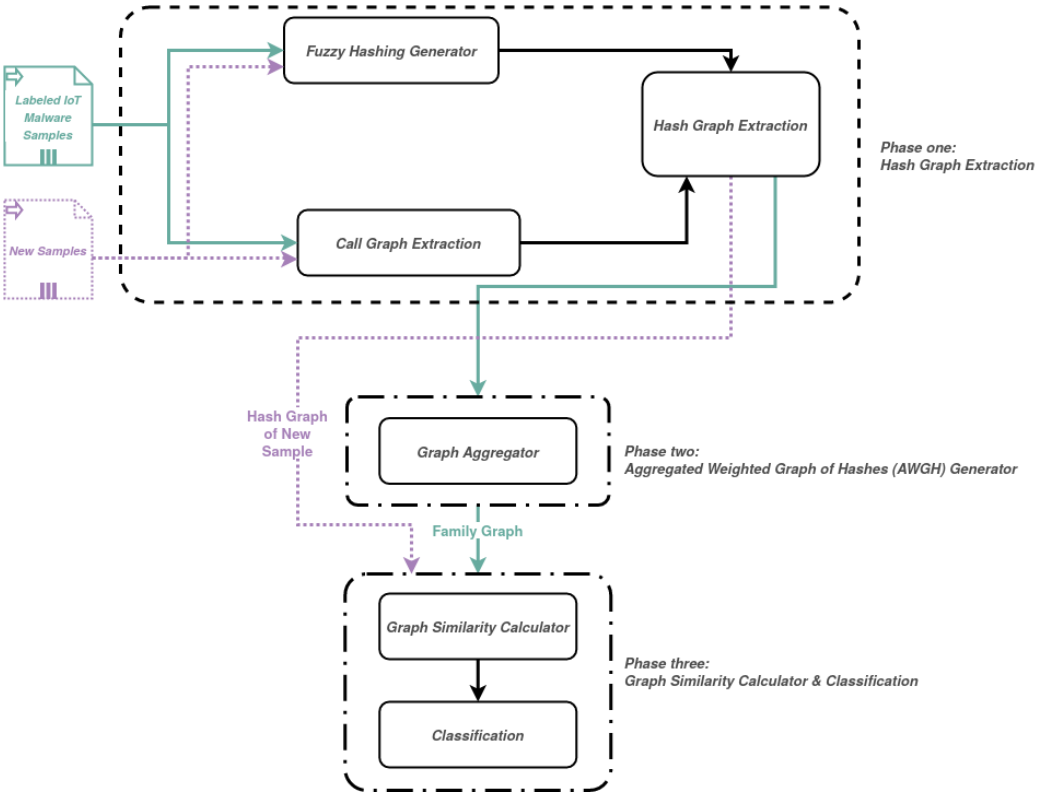


Figure 4.1: Proposed System Overview

- Phase One (Hash Graph Extraction): Labeled IoT malware samples are sent to this generator as the input of two sub-phases of this phase. In this step, the FCGs of all samples and fuzzy hashes of all functions are calculated for further action. Subsequently, each sample's hash graph

will be created by employing the generated FCGs and fuzzy hashes.

- Phase Two (Aggregated Weighted Graph of Hashes (AWGH) Generator): An AWGH has been built upon the generated hash graphs of all samples in a family and represents that family's structure. In the following, we will discuss AWGH more in detail.

- Phase Three (Graph Similarity Calculator & Classification): The graph similarity calculator computes the similarity score between the hash graph of an incoming IoT malware sample and all AWGHs from different families. This similarity score will classify the incoming IoT malware samples into an appropriate IoT malware family.

4.3.1 Aggregated Weighted Graph of Hashes (AWGH)

AWGH or family graph is a weighted graph that is built by merging all hash graphs of samples in an IoT malware family. Hash graph of a sample is like a CFG, in which each node keeps the function's name and the fuzzy hash of that function, unlike the typical CFGs that each node is just the function's name. An AWGH is made by merging all hash graphs of the samples in an IoT malware family and representing that malware family's structure. In AWGH, each node is a set of similar subroutines that have been merged as a supersubroutine. In other words, after generating all hash graphs for each malware's family, those hash graphs of a family will be aggregated in a weighted graph as an AWGH, which is a graph of that family.

4.4 Phase One

In the first phase of our proposed system, Hash Graph Extraction, labelled IoT malware samples are used as inputs to generate FCGs of all samples and fuzzy hashes of all functions. After that, the component of the hash graph extraction creates the hash graph for that IoT sample. Although the generated hash graph structure is the same as FCG, we add the fuzzy hash of that function to each node of the graph. So, nodes represent both the name and fuzzy hash of each function in the hash graph. To be more specific, phase one of the proposed system concludes the following components that we discuss as follow:

1. Fuzzy Hashing Generator: Fuzzy hashing generator receives IoT malware samples (EFL file) as the inputs and extracts all functions for every sample. After that, the fuzzy hashes of all extracted functions are calculated. So, the calculated fuzzy hashes are the output of this part. Thus, we generate the fuzzy hashes of each malware's functions by using the *ssdeep* [1]. *ssdeep* is a tool for computing fuzzy hashes, also called context triggered piecewise hashes (CTPH). We discussed the background about fuzzy hashing and *ssdeep* more specifically in Chapter 2. As our fifth (Con_5) contribution of this thesis, we use fuzzy hashing in our proposed classification framework.

2. Call Graph Extraction: This component is responsible for extracting the function call graphs (FCGs) for all IoT malware samples. For doing

so, we use IDA Pro [60] for generating graphs (CFGs) in this phase by employing the Python code to generate all FCGs automatically.

Function Call Graph (FCG) of a sample are defined as follows:

A FCG is a directed graph $G_S = (V_S, E_S)$, where $V_S = V_S(G)$ represents subroutines (functions) of a sample, and $E_S = E_S(G)$, where $E_S(G) \subseteq V_S(G) \times V_S(G)$, expresses the function calls.

3. Hash Graph Extraction: Hash graph extraction uses two previous steps outputs, the fuzzy hashes of all functions and the FCG. This step is responsible for generating the graph of fuzzy hashes of functions that displays the relationships of functions the same as FCGs. The Hash graph includes the hashes of all "body functions" in each node and the same edges as the FCG. In other words, the result is a graph that nodes are fuzzy hashes of functions, and the edges represent the exact call graph structure of functions in a sample. Therefore, the output of phase one is the hash graphs of the inputs.

   Hash Graph of a sample can be defined as follows:

$$G_H = (V_H, E_H),$$
$$V_H = \{(v, FZH^1(v)) | v \in V_S\},$$
$$E_H \subseteq \{(x, y) | (x, y) \in V_H \times V_H \text{ and } x \neq y\}.$$

[1]Fuzzy hash

4.5 Phase Two

The inputs for this phase are the hash graphs that have been extracted in phase one. In this phase, the Aggregated Weighted Graph of Hashes (AWGH) will be generated for each IoT malware family separately based on the hash graphs of all samples in a family.

Aggregated Weighted Graph of Hashes (AWGH) of a family can be defined as follows:

$$G_F = (V_F, E_F),$$
$$V_F = \text{Set of SuperSubroutines},$$
$$E_F \subseteq \{(x, y) | (x, y) \in V_F \times V_F \text{ and } x \neq y\}.$$

For this step, we start with an empty family and expand it by adding hash graphs of each sample. Assume that we have the family graphs, and we want to add the hash graph of another new sample to this family graph. We compare each node of the sample's hash graph with all of the family graph nodes. If we find a similar node in the family graph, we merge the node from the sample with the family graph node. If not, the new nodes will be added to the family graph. After that, we will update or add the edges to the family graph accordingly. In the end, when we added all the samples to the family, we will have the complete AWGH, which describes the structure of a specific malware family. For aggregating the graphs, we use Python for comparing the similarity between all hash graphs to decide which samples

and nodes can be merged to create the family graph. The Python code has been used to extract the function names, function bodies, and the details of each malware that are accessible by IDA Pro. This piece of code can be used as an IDA Pro plug-in for other related developments.

To sum up, we use Python and IDA Pro to implement our proposed system and make it fully automated to generate a comprehensive graph for each IoT malware family. As our fourth (Con_4) contribution of this research, we created an Aggregated Weighted Graph of Hashes (AWGH) for each IoT malware family, which represents the structure of each malware family.

4.6 Phase Three

In this phase, graph similarity calculator & classification, we will compute the similarity score between each incoming malware hash graph and all graph families (AWGHs). This component is responsible for finding the most similar IoT malware family for the input malware. Although we study some main techniques for calculating the graph similarity in Chapter 1, we use neither of them. In this thesis, since we have some information about the nodes (like names and fuzzy hashes of all functions), this information help compare the graphs more efficiently. Moreover, unlike previous methods for matching the graphs that would ignore some nodes depending on the approach, all nodes (functions) are essential to know which sample can be assigned to which family. So, we compare the nodes from the incoming sample's hash graph to all

family graphs' nodes one by one to find the most similar match.

We can then calculate the similarity score by calculating the percentage of the matched nodes out of all sample nodes. So, the sample will be classified into the family with a higher similarity score than others. As our first (Con_1) contribution of this thesis, our proposed framework classifies IoT malware families based on the AWGHs.

Family Classification of a sample is defined as follows:

For each F in Families,

$$\underset{v \in V_S}{\text{matched}_{\text{F}}}(v) := \{v' | v' \in V_F \ \wedge Sim(v', v) > Threshold \wedge \forall x \in V_F : Sim(v', v) > Sim(x, v)\}$$

$$V_{Matched(F)} = \{v | \underset{v \in V_S}{\text{matched}_{\text{F}}}(v) \neq \varnothing\}$$

$$\text{Assigned Family} = \underset{F \in Families}{\text{argmax}} \frac{|V_{Matched(F)}|}{|V_S|}$$

4.7 Concluding Remarks

In this chapter, we discussed several contributions of this thesis that have been listed before in Chapter 1. First, we discussed the different phases of our graph-based proposed framework to describe each part's tasks to classify the samples (Con_1). Also, we examined how to calculate the similarity between each IoT malware sample and a family in our proposed graph-based approach by creating an Aggregated Weighted Graph of Hashes (AWGH). AWGH

represents the family structure to determine the appointed family (Con_4). Furthermore, we explained how we employ the fuzzy hashing techniques in our proposed IoT malware family classification approach (Con_5).

As our last (Con_6) contribution of this thesis, we explained the details of our proposed fully automated graph-based framework, which classifies IoT malware families each CPU architecture (MIPS, ARM, i386, PowerPC, and AMD64). Chapter 5 will discuss more details about the implementation of the proposed framework.

Chapter 5

Implementation

5.1 Overview

This chapter explains the implementation of the proposed IoT malware family classification framework and provides details about the tools, methods, implementation details of proposed modules, and an architectural overview of the proposed system as well. Our proposed framework has been built upon IDA Pro's (ida-7.2) [60] outputs. IDA Pro is an interactive disassembler that we use in this thesis to generates all required graphs for the IoT samples. All other modules have been developed in Python to be compatible with IDA Pro. The framework's current implementation is illustrated in the module view of the framework, followed by the description of each class's important functions. We employ class diagrams to illustrate the proposed framework's structure by showing the system's classes, their attributes, and

the relationships among objects. The module view represents the structure that all components of the framework are united as a program. In the following sections, we discuss the proposed system's module view to show how the proposed system decomposes into manageable software units.

5.2 Module View

The module view displays the way that the proposed framework is decomposed. Modules are the system's implementation units that provide a coherent unit of functionality that make the elements of the module view.
Figure 5.1 illustrates the module view of the proposed framework in the UML class diagram and the description of each class as well.

5.2.1 GDL & Assembly Generator

This module generates assembly files and Graph Description Language (GDL) files for all input samples. GDL is a file that contains information that describes the function call graphs that we use in this thesis. All IoT malware samples with different supporting architecture, including MIPS, i386, and ARM are fed into this IDA Pro module to generate the assembly files and GDL files, which creates the main inputs for the proposed system.

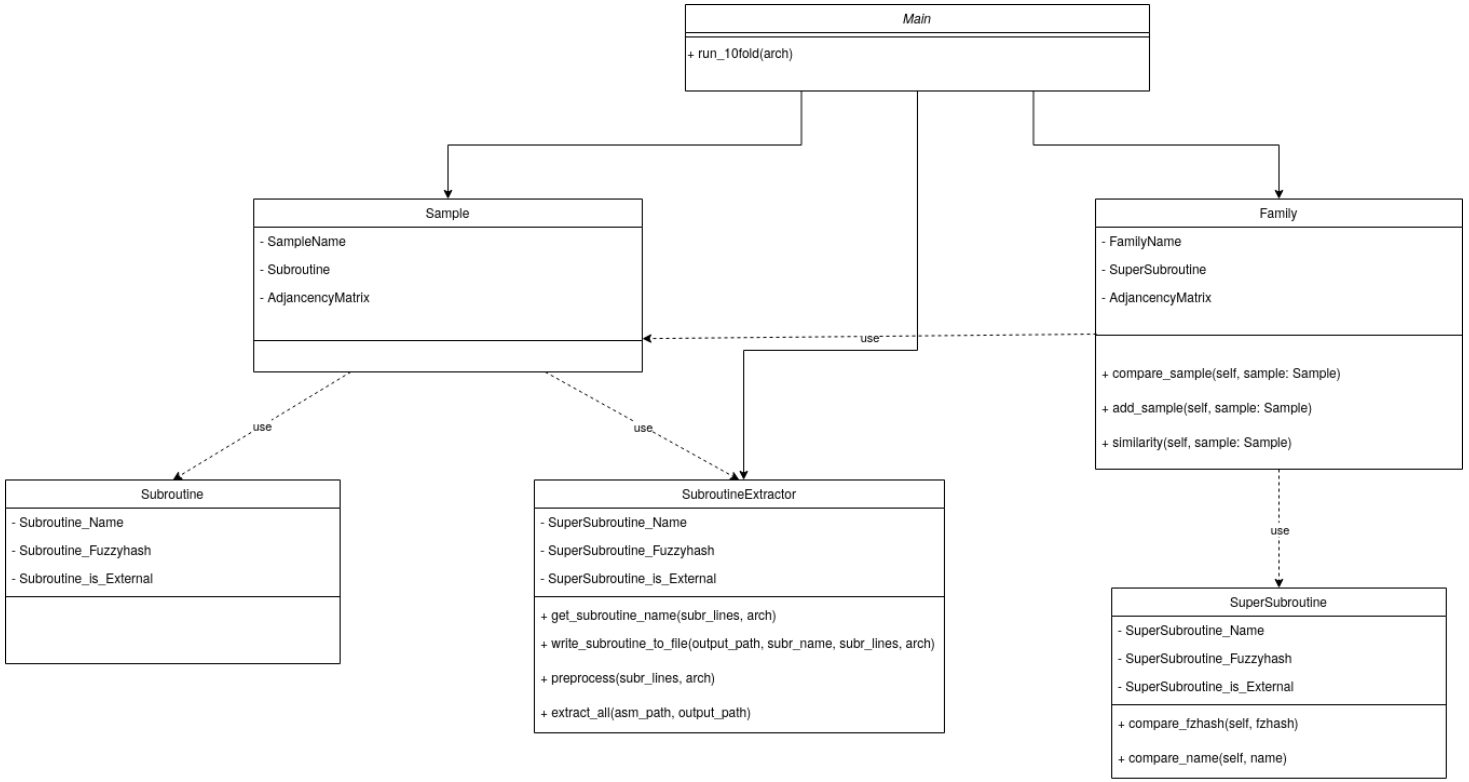


Figure 5.1: Proposed System Class Diagram

5.2.2 *Sample* Class

This class represents an IoT malware sample that contains its name (hash), subroutines, and assigned fuzzy hashes, and the adjacency matrix of the FCG. This class generates the hash graph of all samples by using the generated FCGs and fuzzy hashes of subroutines.

- Subroutines: The class of *Subroutine Extractor* provides a list of all subroutines of the sample. After generating each subroutine, we use a fuzzy hashing tool named *ssdeep* in the Python code [1] to compute the fuzzy hash of each generated subroutine (function). By using the fuzzy hash of subroutines, we can compare the similarity of subroutines in the next step, which is creating a *Family* object.

  There are a few external subroutines in the IoT malware codes. As we know, the name of the external subroutine is enough for the comparison because external subroutines are named in a standard way that keeps them unique.

- Adjacency Matrix: The samples' adjacency matrix shows the relationships between the subroutines to see which one of them is "caller" or "callee". Figure 5.2 displays a small part of a sample's adjacency matrix. For example, subroutine 8 calls subroutine 7, and subroutine 12 calls subroutine 10. Note that the assigned number of subroutines are based on their order in GDLs.

	7	8	9	10	11	12	13	14
0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
1	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
2	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
3	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
4	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
5	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
6	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
7	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
8	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
9	1.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
10	1.00000	1.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.0000
11	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.0000
12	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.0000
13	0.00000	0.00000	0.00000	1.00000	1.00000	0.00000	0.00000	0.0000
14	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
15	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
16	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000

Figure 5.2: Small Part of the Adjacency Matrix of one IoT Malware sample

5.2.3 *Subroutine* Class

In this class, we define the *Subroutine* class to use in other parts of the implementation. The subroutines have been defined by their name, fuzzy hash, and whether they are external or internal. We use this class in "Sample Graph Generator" for extracting the subroutines for the samples.

5.2.4 *Subroutine Extractor* Module

The *subroutine extractor* is responsible for extracting all subroutine from the samples' assembly files and saving them by their name for each sample for further analysis. The assembly code generated by IDA Pro contains some

parts (for example, comments) that have no significance in comparing two subroutines. Before storing these subroutines, we perform a preprocessing on them by using *preprocess* function in this module. The assembly codes of the samples are clean to express essential and meaningful differences or similarities.

5.2.5 *SuperSubroutine* Class

In this class, we define the super-subroutines to use in other parts of the implementation. A super-subroutine will be created when we merge two similar subroutines of two different samples from a particular family. Super-subroutines have been defined by a list of names and a list of different subroutine's fuzzy hashes merged. Also, there is a flag that shows whether the super-subroutine is external or internal.

5.2.6 *Family* Class

This class represents the family by using the attributes, including the family name, super-subroutines, and the family graph's adjacency matrix. In this class, all sample graphs that have been generated in "Sample Graph Generator" will be aggregated to create the graph for each IoT malware family. In this graph, nodes are super-subroutines containing fuzzy hashes of all the merged subroutines, and edges are weighted to show how many times each of the nodes has been called by others. We call this family graph an Aggregated

Weighted Graph of Hashes (AWGH). These are the attribute for this class:

- Super-subroutines: For each family, we keep a list of super-subroutines that is a result of merging different samples, each with a list of subroutines.
- Adjacency Matrix: The families' adjacency matrix shows the relationships between the super-subroutines to see which group of super-subroutines are "caller" or "callee".

5.2.7 *Main* Module

The entire process of the proposed system is controlled by the *Main* starting from generating the assembly files and GDLs of the samples to classifying the incoming samples.

5.3 Concluding Remarks

This chapter presents the design and implementation of the proposed system. It provides important details of implementation, such as describing the class and important functions and role of each part to make this proposed system. In this chapter, we draw the system's module view, which employs the UML class diagram, showing the system's implementation modules.

Chapter 6

Experiments & Results

6.1 Overview

This chapter starts by explaining the malware samples and dataset that we use in this thesis. To demonstrate the proposed system's capability, we provide the evaluation sections for each experiment regarding the data. Since our proposed system is a pioneering approach in the IoT landscape, there is no similar state-of-art piece of work for comparative evaluation. For example, as we discussed in Chapter 3, Su *et al.* (2018) represent an image-based classification system that classifies the samples into two major IoT families (Mirai & Gafgyt) with an accuracy of 81.8% [67]. In this chapter, first, we review the dataset. After that, we provide results for each of our experiments to see the proposed system's effectiveness.

6.2 Dataset

All the samples used in this thesis are downloaded from VirusTotal [4]. VirusTotal gathers scan results from 75 different antivirus or antimalware systems (e.g. NOD32, McAfee, Kaspersky, etc.), and based on these scan results; we determine the label of the samples. Each malware sample has an Executable and Linkable Format (ELF) file of the malware and a *.json* file which contains some information like scan results, supporting CPU architecture, etc. It is worth mentioning that in some cases, there may be a disagreement in the samples' labels that have been detected by VirusTotal's sources (antivirus or antimalware systems). For example, for a particular sample, some of the antiviruses detect the sample as a Gafgyt, but some other antiviruses detect the same sample as Hajime. Here for labelling the samples, we simply consider the most frequent label. However, keep in mind that we cannot be 100% certain about the label of each sample because of these disagreements. Table 6.1 displays the distribution of ELF samples that we use in this thesis based on the supporting CPU architectures. In this dataset, ARM, and MIPS are the most supported architectures of malware. Moreover, Gafgyt and Mirai have the most number of samples for each family. There are several families of IoT malware such as Hajime, Tsunami, MrBlack, Dofloo, and Aidra; the number of samples in each architecture group is not enough for training our system. Thus, we assume all of them as one group named "others" just to test the proposed system.

Table 6.1: Distribution of malware samples based on the CPU architecture

Architecture	Total	Mirai	Gafgyt	Hajime	Tsumami	MrBlack	Dofloo	Aidra
ARM	464	234	136	4	6	31	53	0
MIPS R3000	322	148	148	3	20	2	1	0
i386	214	67	97	0	11	16	23	0
PowerPC	126	67	56	0	3	0	0	0
AMD x86-64	84	10	61	0	7	2	1	3

As we mentioned before, all FCGs are generated by IDA Pro in this thesis, but IDA Pro cannot generate CFGs for a few numbers of IoT malware in all families. So, this is the reason that the total number of analyzed malware in the next section and total samples in Table 6.1 are a bit different. In this thesis, we use 10-fold cross-validation to estimate the effectiveness of the system on the data. In other words, the data samples are randomly divided into ten groups, and each time, we keep one of these groups for testing and train our system with the other nine groups. Then we report the average values (for precision, recall, etc.) of these ten experiments.

6.3 System Evaluation

To evaluate the effectiveness of the proposed system, we have performed several experiments. As mentioned before, we perform these experiments

separately for each CPU architecture (MIPS, ARM, i386, PowerPC, and AMD64). Also, because the number of samples in families other than Mirai or Gafgyt is very low, we only create family graphs for these two families. In the following sections, we present the results for each CPU architecture.

6.3.1 MIPS

In the first experiment for MIPS architecture, we only use the samples from malware families Mirai and Gafgyt. We use 10-fold cross-validation in our experiment. Table 6.2 represents the confusion matrix for this experiment.

Table 6.2: Results of classifying MIPS samples into two groups of Mirai and Gafgyt

		Predicted Family	
		Mirai	Gafgyt
Family	Mirai	122	1
	Gafgyt	3	130

In total, our proposed system detects 122 Mirai samples correctly, except for one sample that has been classified as Gafgyt. For the Gafgyt category, three samples are marked as Mirai, and 130 samples have been detected correctly. It is a two-class classification that describes the proposed method's functionality between two popular IoT malware family. Table 6.3 displays the measured evaluation metrics for our proposed system. The results show that the system classifies the samples effectively.

Table 6.3: Evaluation metrics for MIPS architecture

	precision	recall	F-1 Measure
Mirai	%97.60	%99.18	%98.38
Gafgyt	%99.23	%97.74	%98.48

For the second experiment, we use samples from other families (e.g. Hajime, MrBlack, etc.) but only for testing purposes. Also, in this experiment, we define a minimum threshold for similarity. It means we only consider a test sample as a member of a family, only if their similarity is above a threshold. If not, we do not detect the sample as a member of that particular family. For this experiment, we chose the threshold 70%. Table 6.4 shows the confusion matrix for this experiment.

Table 6.4: Results of classifying MIPS samples into two groups of Mirai and Gafgyt when threshold is 70%

		Predicted Family		
		Mirai	Gafgyt	N/A
Actual Family	Mirai	100	0	23
	Gafgyt	0	127	6
	Others	30	50	160

Table 6.5 illustrates the measured evaluation metric for the experiment above. There are several reasons that the results in Table 6.5 are not as good as Table 6.3. As we mentioned before, VirusTotal cannot label the samples accurately due to the difference between the antiviruses detection results. Therefore, some of the samples that are groped as “others” might actually

belong to one of these families. As we mentioned before, VirusTotal cannot label the samples accurately due to the difference between the antiviruses detection results. Therefore, some of the samples that are groped as "others" might actually belong to one of these families. Moreover, some malware families are very similar to each other or might even be derived from another family. For example, the Hajime malware family is very similar to the Gafgyt malware family in structure. In addition, some of the samples with lower similarity to their respective family could not be detected by introducing the threshold.

Table 6.5: Evaluation metrics for MIPS architecture when the threshold is 70%

	precision	recall	F-1 Measure
Mirai	%76.92	%81.30	%79.05
Gafgyt	%71.75	%95.48	%81.93

Figure 6.1 shows the similarity of each test sample to each of the trained families. Since we only have two families in this architecture (Mirai and Gafgyt), we show it as a 2D diagram, in which the *x axis* shows the similarity score with the Mirai family, and the *y axis* shows the similarity with Gafgyt family. As you can see, many of the Mirai Samples (orange dots) are very close to the right side of the diagram. Many of the Gafgyt samples are at the top of the diagram, which means a high similarity score with the Gafgyt family.

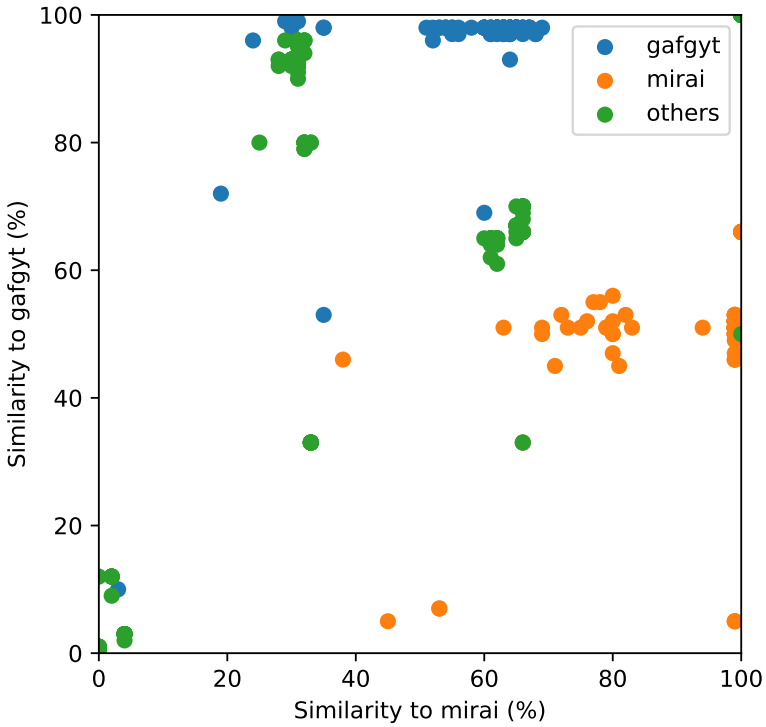


Figure 6.1: Classification on MIPS samples

6.3.2 ARM

In this section, we present the results for the ARM architecture. In the first experiment, similar to what we did for MIPS, we only use the samples from malware families Mirai and Gafgyt. Table 6.6 represents the confusion matrix for this experiment. Our proposed system detects all Mirai samples correctly, and for the Gafgyt category, four samples are marked as Mirai, and 111 samples have been detected correctly.

Table 6.6: Results of classifying ARM samples into two groups of Mirai and Gafgyt

		Predicted Family	
		Mirai	Gafgyt
Family	Mirai	209	0
	Gafgyt	4	111

Table 6.7 illustrates the measured evaluation metrics for ARM architecture. The results show that the system classifies the samples effectively.

Table 6.7: Evaluation metrics for ARM architecture

	precision	recall	F-1 Measure
Mirai	%98.12	%100	%99.058
Gafgyt	%100	%96.52	%98.20

For the second experiment in the ARM category, we use samples from other families (e.g. Hajime, MrBlack, etc.) but only for testing purposes, the same as a similar experiment for the MIPS. Also, we use a minimum similarity threshold to classify the samples above the minimum, and like before, we select 70% for the threshold. Table 6.8 displays the confusion matrix for this experiment. Moreover, Table 6.9 shows the measured evaluation metric for the experiment above. As we discussed in the previous section for a similar experiment for MIPS, labelling problems, the similarity between families, and defining the threshold are the main reasons that the results in Table 6.9 are not as good as Table 6.7.

Table 6.8: Results of classifying ARM samples into two groups of Mirai and Gafgyt when threshold is 70%

		Predicted Family		
		Mirai	Gafgyt	N/A
Actual Family	Mirai	161	0	48
	Gafgyt	3	109	3
	Others	30	10	330

Table 6.9: Evaluation metrics for ARM architecture when the threshold is 70%

	precision	recall	F-1 Measure
Mirai	%82.98	%77.03	%79.90
Gafgyt	%91.59	%94.78	%93.16

Figure 6.2 describes the similarity of each test sample to each of the trained families. Since we only have two families in this architecture (Mirai and Gafgyt), we display it as a 2D diagram, in which the *x axis* represents the similarity score with the Mirai family, and the *y axis* represents the similarity with the Gafgyt family.

As you can see, many of the Mirai Samples (orange dots) are very close to the right side of the diagram, and also many of the Gafgyt samples are at the top of the diagram, which means a high similarity score with the Gafgyt family almost same as Figure 6.1 for MIPS samples in the previous section.

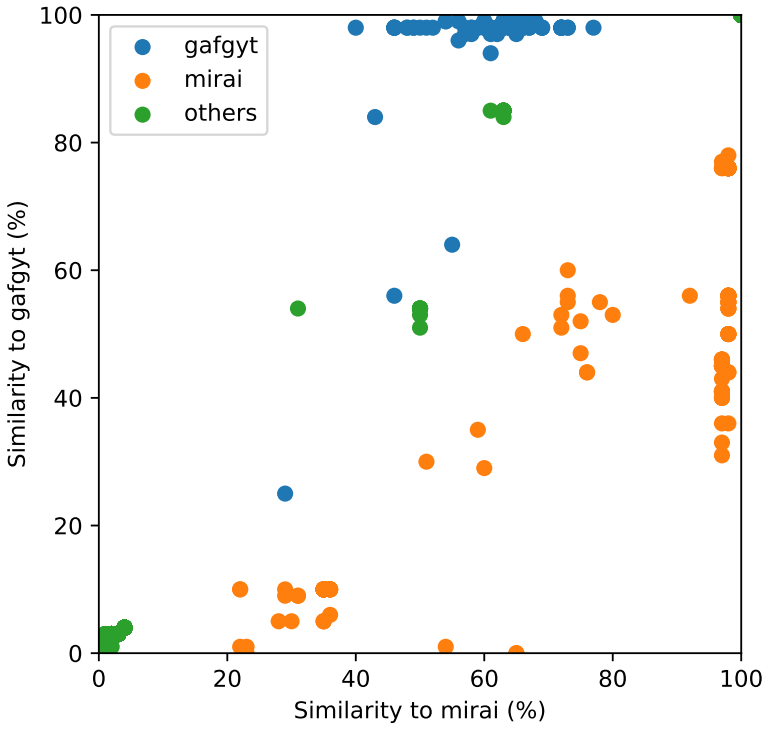


Figure 6.2: Classification on ARM samples

6.3.3 Intel (i386)

In this section, we present the results for i386 architecture. Table 6.10 illustrates the confusion matrix for only using the Mirai and Gafgyt samples in this CPU architecture. Our proposed system detects all Mirai samples correctly, and for the Gafgyt category, only one sample is detected as Mirai, and 82 samples have been marked correctly.

Table 6.10: Results of classifying i386 samples into two groups of Mirai and Gafgyt

		Predicted Family	
		Mirai	Gafgyt
Family	Mirai	39	0
	Gafgyt	1	82

Table 6.11 represents the measured evaluation metrics for i386 architecture. The results illustrate that the system groups the samples effectively.

Table 6.11: Evaluation metrics for i386 architecture

	precision	recall	F-1 Measure
Mirai	%97.5	%100	%98.73
Gafgyt	%100	%98.79	%99.39

Table 6.12 shows the confusion matrix for the second experiment in the i386 category. In this step, a minimum similarity threshold (70%) has been defined to classify the samples within Mirai, Gafgyt, and other families.

Table 6.12: Results of classifying i386 samples into two groups of Mirai and Gafgyt when threshold is 70%

		Predicted Family		
		Mirai	Gafgyt	N/A
Actual Family	Mirai	39	0	0
	Gafgyt	0	82	1
	Others	0	49	331

Table 6.13 shows the measured evaluation metric for the second experiment in i386 architecture. All three metrics for Mirai in this architecture is equal

to %100. As we discussed in the previous section for a similar experiment for MIPS, labelling problems, the similarity between families, and defining the threshold are the main reasons that the results in Table 6.13 are not as good as Table 6.11.

Table 6.13: Evaluation metrics for i386 architecture when the threshold is 70%

	precision	recall	F-1 Measure
Mirai	%100	%100	%100
Gafgyt	%62.59	%98.79	%76.63

Figure 6.3 shows that all of the Mirai Samples (orange dots) are close to the right side of the diagram, representing their similarity to the Mirai family is high. Also, most of the Gafgyt samples are at the top of the diagram, which means a high similarity score with the Gafgyt family. Green dots represent "others" family, and their similarities to the Mirai and Gafgyt are defined in this diagram.

Since we only have two families in this architecture (Mirai and Gafgyt), we display it as a 2D diagram, in which the *x axis* represents the similarity score with the Mirai family, and the *y axis* represents the similarity with the Gafgyt family.

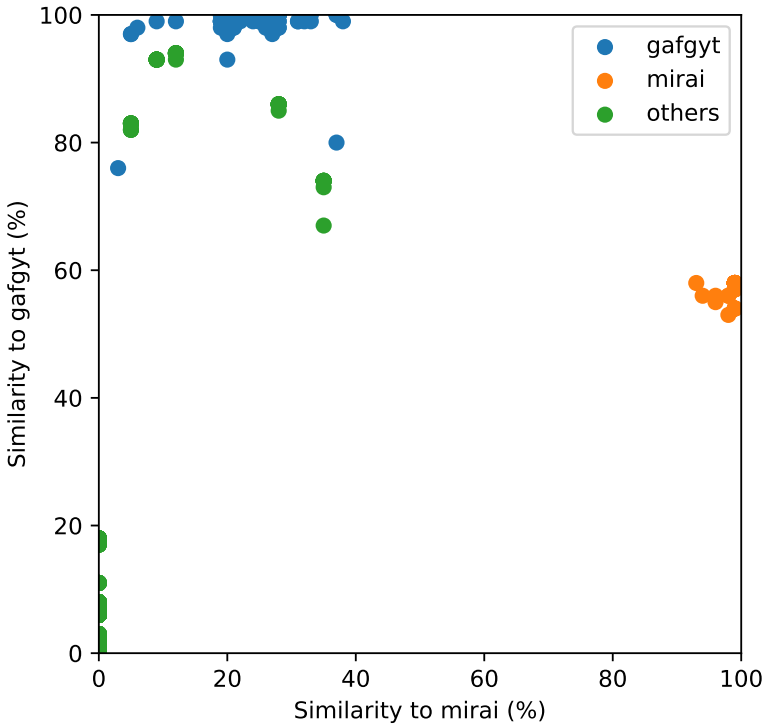


Figure 6.3: Classification on i386 samples

6.3.4 PowerPC

In this part, we represent the results for PowerPC architecture. Table 6.14 displays the confusion matrix for only using the Mirai and Gafgyt samples in PowerPC architecture. Our proposed system detects all 54 Mirai samples correctly, and for the Gafgyt category, only 3 out of 49 samples are defined as Mirai.

Table 6.14: Results of classifying PowerPC samples into two groups of Mirai and Gafgyt

		Predicted Family	
		Mirai	Gafgyt
Family	Mirai	54	0
	Gafgyt	3	46

Table 6.15 represents the measured evaluation metrics for PowerPC architecture. The results illustrate that the system groups the samples effectively.

Table 6.15: Evaluation metrics for PowerPC architecture

	precision	recall	F-1 Measure
Mirai	%94.73	%100	%97.29
Gafgyt	%100	%93.87	%96.84

Table 6.16 shows the confusion matrix for the second experiment in the PowerPC category. In this experiment, a minimum similarity threshold (70%) has been determined to classify the samples within Mirai, Gafgyt, and other families (e.g. Hajime, MrBlack, etc.).

Table 6.16: Results of classifying PowerPC samples into two groups of Mirai and Gafgyt when threshold is 70%

		Predicted Family		
		Mirai	Gafgyt	N/A
Actual Family	Mirai	52	0	2
	Gafgyt	3	44	2
	Others	0	9	11

Table 6.17 represents the second experiment's measured evaluation metric in the PowerPC category when the threshold is 70%.

Table 6.17: Evaluation metrics for PowerPC architecture when the threshold is 70%

	precision	recall	F-1 Measure
Mirai	%94.54	%96.29	%95.41
Gafgyt	%83.01	%89.79	%86.27

Figure 6.4 shows the similarity of the samples to the Mirai or Gafgyt families.

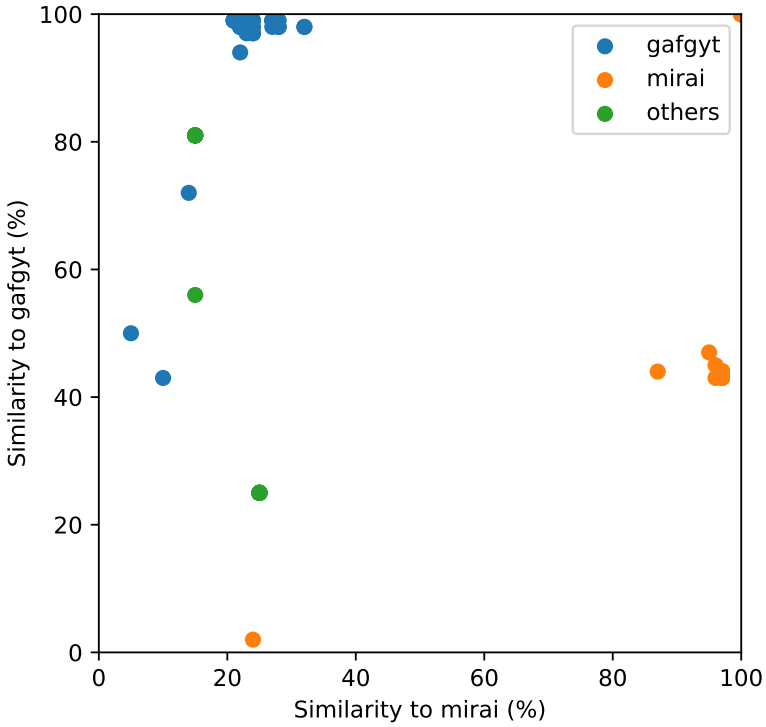


Figure 6.4: Classification on PowerPC samples

6.3.5 AMD x86-64

In AMD x86-64 architecture, we only have enough samples for one family (Gafgyt) to train it. So, we use samples from other families (e.g. Mirai, Hajime, MrBlack, etc.) for testing purposes and perform our experiment using 10-fold cross-validation. Table 6.18 represents the confusion matrix for this experiment. Unlike the previous experiments which are two-class classification, this experiment is a one-class classification. In total, our proposed system marks 52 Gafgyt samples correctly, except for one sample that has been classified as Not Applicable (N/A). For the "others" category, 170 samples have been defined correctly, and ten samples are detected as Gafgyt.

Table 6.18: Results of classifying AMD x86-64 samples into two groups of Gafgyt and others when threshold is 70%

		Predicted Family	
		Gafgyt	N/A
Actual Family	Gafgyt	52	1
	Others	10	170

As we mentioned before, VirusTotal cannot label the samples accurately due to the difference between the antiviruses detection results. Therefore, some of the samples that are groped as Gafgyt might actually belong to one of the "others" families. In this experiment, we define a minimum threshold (70%) for similarity. It means we only consider a test sample as a family member, only if their similarity is above a threshold. Moreover, some malware families are very similar to each other or might even be derived from another family.

For example, the Hajime malware family is very similar to the Gafgyt malware family in structure. Besides, some of the samples with lower similarity to their respective family could not be detected by introducing the threshold. Table 6.19 displays the measured evaluation metric for the experiment above.

Table 6.19: Evaluation metrics for AMD x86-64 architecture when the threshold is 70%

	precision	recall	F-1 Measure
Gafgyt	%83.87	%98.11	%90.43

6.4 Conclusion Remarks

This chapter evaluates our proposed framework in terms of effectiveness by testing it on the VirusTotal dataset. Moreover, we provide a comparative analysis with different types of IoT malware regarding the types of CPU architectures. We perform two sorts of experiments on the dataset. In the first experiment for each architecture, we only use the samples from malware families Mirai and Gafgyt. And after that, we perform 10-fold cross-validation. The first experimental results for each CPU architecture demonstrate the proposed method's functionality between two popular IoT malware families. For the second experiment, we also use other families, including Hajime, Mr-Black, etc. but only for testing purposes. In this experiment, we specify a minimum threshold of 70% for similarity. So, a test sample is considered a

family member when their similarity is more than the threshold. According to some reasons, the results of the first experiment are much better than the second one. As we discussed before, the first reason is the inaccuracy of VirusTotal's labels that makes some of the samples are known as "others" might really be linked to one of the families like Mirai or Gafgyt. Furthermore, the structures of some malware families are very similar to each other. As an instance, the Hajime malware family structure is very similar to the Gafgyt malware's structure. Besides, defining the threshold causes not to detect some samples with lower similarity.

Table 6.20 shows the summary of the results for both experiments above.

Table 6.20: Summary of the results

Arch.	Expe.	Mirai			Gafgyt		
		Pre.	Rec.	F-1	Pre.	Rec.	F-1
MIPS	First	97.60%	99.18%	98.38%	99.23%	97.74%	98.48%
	Second	76.92%	81.30%	79.05%	71.75%	95.48%	81.93%
ARM	First	98.12%	100%	99.058%	100%	96.52%	98.20%
	Second	82.98%	77.03%	79.90%	91.59%	94.78%	93.16%
i386	First	97.5%	100%	98.73%	100%	98.79%	99.39%
	Second	100%	100%	100%	62.59%	98.79%	76.63%
PowerPC	First	94.73%	100%	97.29%	100%	93.87%	96.84%
	Second	94.54%	96.29%	95.41%	83.01%	89.79%	86.27%
x86-64	First	N/A	N/A	N/A	83.87%	98.11%	90.43%
	Second	N/A	N/A	N/A	N/A	N/A	N/A

Chapter 7

Conclusions & Future Work

7.1 Conclusions

Lack of research on the IoT malware family has motivated us to build an automated IoT malware family classification framework. We found most of the current studies focus on classifying between malicious or non-malicious IoT software and detecting among Android or IoT samples instead of distinguishing between different malware families. For the first time in the IoT landscape, we proposed a fully automated graph-based classification framework in Python by employing fuzzy hashing. In this thesis, we use IDA Pro (*ida-7.2*) [60] for the FCGs generation, and *ssdeep* [3] for computing the fuzzy hashes to develop our proposed framework. By employing the combination of FCGs and fuzzy hashing, we introduced the Aggregated Weighted Graph of Hashes (AWGH) that can be generated for each IoT malware fam-

ily to reflect that family's structure. The graph of every single IoT malware is compared to all AWGHs from different families to find the best match. For estimating the effectiveness of the proposed framework, we use 10-fold cross-validation on the VirusTotal dataset [4] and conduct the comparative analysis for different types of IoT malware with different CPU architectures (MIPS, ARM, i386, PowerPC, and AMD64). The results illustrate the proposed classification system is effective.

Furthermore, we reviewed the literature toward the IoT malware detection/-classification regarding the types of features to propose a novel taxonomy. Our proposed taxonomy displays the different groups of features toward the kinds of approaches. Thus, we provide a survey of the features that previous studies have used in their IoT malware detection approaches.

7.2 Future Work

- Parameter tuning: In this thesis, all subroutines with the same name that have 50% or more similarity scores can be merged as a super-subroutine, and if the name is not the same, the required similarity score is 70%. We defined these thresholds by trying different values. However, a more thorough parameter tuning could be done to reach the optimal value for these thresholds.

- Apply different graph similarity algorithms: Different algorithms for calculating the similarity between graphs can be used instead of our

simple technique to compare the efficiency and effectiveness of the system in various scenarios.

- Employ a combination of CFG and FCG: Our proposed framework can generate both CFG and FCG So, the classification will be evaluated by combining both graphs as input to the framework to compare the effectiveness to the current solution.

- Employ a combination of graph-based features and current system: Adding graph-based features that describe the individual aspect of binaries to get mixed with the family graph that we represent in this thesis can bring a group of new results for more discussions.

- Experiment with more diverse samples of IoT malware with various types of CPU architecture: Analyzing more recent samples can evaluate the system more sophisticated than now. Additionally, having enough diverse samples with different kinds of architecture help to train a comprehensive classification system.

Bibliography

[1] *Fuzzy hashing program*, Available at `https://ssdeep-project.github.io/ssdeep/index.html` [Online; Accessed Nov. 2020].

[2] *Public cyberiocs repository*, Available at `https://freeiocs.cyberiocs.pro/` [Online; Accessed Nov. 2020].

[3] *ssdeep project*, Available at `https://github.com/ssdeep-project/ssdeep` [Online; Accessed Nov. 2020].

[4] *Virustotal-free online virus, malware and url scanner*, Available at `https://www.virustotal.com` [Online; Accessed Nov. 2020].

[5] Ahmed Abusnaina, Aminollah Khormali, Hisham Alasmary, Jeman Park, Afsah Anwar, Ulku Meteriz, and Aziz Mohaisen, *Breaking graph-based iot malware detection systems using adversarial examples: poster*, Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks, 2019, pp. 290–291.

[6] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash, *Internet of things: A survey on enabling technologies, protocols, and applications*, IEEE communications surveys & tutorials **17** (2015), no. 4, 2347–2376.

[7] Hisham Alasmary, Afsah Anwar, Jeman Park, Jinchun Choi, DaeHun Nyang, and Aziz Mohaisen, *Graph-based comparison of iot and android malware*, International Conference on Computational Social Networks, Springer, 2018, pp. 259–272.

[8] Hisham Alasmary, Aminollah Khormali, Afsah Anwar, Jeman Park, Jinchun Choi, Ahmed Abusnaina, Amro Awad, DaeHun Nyang, and Aziz Mohaisen, *Analyzing and detecting emerging internet of things malware: a graph-based approach*, IEEE Internet of Things Journal **6** (2019), no. 5, 8977–8988.

[9] Hisham Alasmary, Aminollah Khormali, Afsah Anwar, Jeman Park, Jinchun Choi, DaeHun Nyang, and Aziz Mohaisen, *Poster: Analyzing, comparing, and detecting emerging malware: A graph-based approach*, (2019).

[10] Mohannad Alhanahnah, Qicheng Lin, Qiben Yan, Ning Zhang, and Zhenxiang Chen, *Efficient signature generation for classifying cross-architecture iot malware*, 2018 IEEE Conference on Communications and Network Security (CNS), IEEE, 2018, pp. 1–9.

[11] Arghire, *Iot botnet used in website hacking attacks*, 2017, Available at `https://www.securityweek.com/iot-botnet-used-website-used-website-hacking-attacks` [Online; Accessed Nov. 2020].

[12] Amin Azmoodeh and Ali Dehghantanha, *Detecting crypto-ransomware in iot networks based on energy consumption footprint*, Journal of Ambient Intelligence and Humanized Computing (2017), 1–12.

[13] Amin Azmoodeh, Ali Dehghantanha, and Kim-Kwang Raymond Choo, *Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning*, IEEE transactions on sustainable computing (2018).

[14] Amin Azmoodeh, Ali Dehghantanha, Mauro Conti, and Kim-Kwang Raymond Choo, *Detecting crypto-ransomware in iot networks based on energy consumption footprint*, Journal of Ambient Intelligence and Humanized Computing **9** (2018), no. 4, 1141–1152.

[15] Joel W Branch, Chris Giannella, Boleslaw Szymanski, Ran Wolff, and Hillol Kargupta, *In-network outlier detection in wireless sensor networks*, Knowledge and information systems **34** (2013), no. 1, 23–54.

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

[17] Vincenzo Carletti, *Exact and inexact methods for graph similarity in structural pattern recognition phd thesis of vincenzo carletti.*, Ph.D. thesis, 2016.

[18] David Reinsel Carrie MacGillivray, *Worldwide global datasphere iot device and data forecast*, 2019, Available at `https://www.idc.com/` [Online; Accessed Nov. 2020].

[19] Luca Caviglione, Mauro Gaggero, Jean-François Lalande, Wojciech Mazurczyk, and Marcin Urbański, *Seeing the unseen: revealing mobile malware hidden communications via energy consumption and artificial intelligence*, IEEE Transactions on Information Forensics and Security **11** (2015), no. 4, 799–810.

[20] Kai-Chi Chang, Raylin Tso, and Min-Chun Tsai, *Iot sandbox: to analysis iot malware zollard*, Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing, ACM, 2017, p. 4.

[21] Catalin Cimpanu, *There'sa 120,000-strong iot ddos botnet lurking around*, 2016, Available at `https://news.softpedia.com/news/there-s-a-120-000-strong-iot-ddos-botnet-lurking-around-507773.shtml` [Online; Accessed Nov. 2020].

[22] Andrei Costin and Jonas Zaddach, *Iot malware: Comprehensive survey, analysis framework and case studies*, BlackHat USA (2018).

[23] Hamid Darabian, Ali Dehghantanha, Sattar Hashemi, Sajad Homayoun, and Kim-Kwang Raymond Choo, *An opcode-based technique for polymorphic internet of things malware detection*, Concurrency and Computation: Practice and Experience **32** (2020), no. 6, e5173.

[24] Michele De Donno, Nicola Dragoni, Alberto Giaretta, and Angelo Spognardi, *Analysis of ddos-capable iot malwares*, 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), IEEE, 2017, pp. 807–816.

[25] Rohan Doshi, Noah Apthorpe, and Nick Feamster, *Machine learning ddos detection for consumer internet of things devices*, 2018 IEEE Security and Privacy Workshops (SPW), IEEE, 2018, pp. 29–35.

[26] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel, *A survey on automated dynamic malware-analysis techniques and tools*, ACM computing surveys (CSUR) **44** (2008), no. 2, 1–42.

[27] Xuan Feng, Qiang Li, Haining Wang, and Limin Sun, *Acquisitional rule-based engine for discovering internet-of-things devices*, 27th {USENIX} Security Symposium ({USENIX} Security 18), 2018, pp. 327–341.

[28] Tatikayala Sai Gopal, Mallesh Meerolla, G Jyostna, P Reddy Lakshmi Eswari, and E Magesh, *Mitigating mirai malware spreading in iot environment*, 2018 International Conference on Advances in Computing,

Communications and Informatics (ICACCI), IEEE, 2018, pp. 2226–2230.

[29] Waylon Grange, *Hajime worm battles mirai for control of the internet of things*, Symantec Blog, April (2017).

[30] Hamed HaddadPajouh, Ali Dehghantanha, Raouf Khayami, and Kim-Kwang Raymond Choo, *A deep recurrent neural network based approach for internet of things malware threat hunting*, Future Generation Computer Systems **85** (2018), 88–96.

[31] Shuang Hao, Nadeem Ahmed Syed, Nick Feamster, Alexander G Gray, and Sven Krasser, *Detecting spammers with snare: Spatio-temporal network-level automatic reputation engine.*, USENIX security symposium, vol. 9, 2009.

[32] Ragib Hasan, Nitesh Saxena, Tzipora Haleviz, Shams Zawoad, and Dustin Rinehart, *Sensing-enabled channels for hard-to-detect command and control of mobile devices*, Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, ACM, 2013, pp. 469–480.

[33] Mehadi Hassen and Philip K Chan, *Scalable function call graph-based malware classification*, Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, 2017, pp. 239–248.

[34] Dustin Hurlbut, *Fuzzy hashing for digital forensic investigators*, 2009, Available at `https://ad-pdf.s3.amazonaws.com/Fuzzy_Hashing_for_Investigators.pdf` [Online; Accessed Nov. 2020].

[35] Ryoichi Isawa, Tao Ban, Ying Tie, Katsunari Yoshioka, and Daisuke Inoue, *Evaluating disassembly-code based similarity between iot malware samples*, 2018 13th Asia Joint Conference on Information Security (AsiaJCIS), IEEE, 2018, pp. 89–94.

[36] Qi Jing, Athanasios V Vasilakos, Jiafu Wan, Jingwei Lu, and Dechao Qiu, *Security of the internet of things: perspectives and challenges*, Wireless Networks **20** (2014), no. 8, 2481–2501.

[37] Georgios Kambourakis, Constantinos Kolias, and Angelos Stavrou, *The mirai botnet and the iot zombie armies*, MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM), IEEE, 2017, pp. 267–272.

[38] Michael Kassner, *ssdeep project - fuzzy hashing program*, 2011, Available at `https://www.techrepublic.com/blog/it-security/fuzzy-hashing-helps-researchers-spot-morphing-malware/` [Online; Accessed Nov. 2020].

[39] Kris Kendall and Chad McMillan, *Practical malware analysis*, Black Hat Conference, USA, 2007, p. 10.

[40] Danai Koutra, Ankur Parikh, Aaditya Ramdas, and Jing Xiang, *Algorithms for graph similarity and subgraph matching*, Proc. Ecol. Inference Conf, vol. 17, 2011.

[41] Brian Krebs, *Hacked cameras, dvrs powered today's massive internet outage*, Krebs on Security (2016).

[42] Ayush Kumar and Teng Joon Lim, *Early detection of mirai-like iot bots in large-scale networks through sub-sampled packet traffic analysis*, Future of Information and Communication Conference, Springer, 2019, pp. 847–867.

[43] Yuping Li, Sathya Chandran Sundaramurthy, Alexandru G Bardas, Xinming Ou, Doina Caragea, Xin Hu, and Jiyong Jang, *Experimental study of fuzzy hashing in malware clustering analysis*, 8th Workshop on Cyber Security Experimentation and Test ({CSET} 15), 2015.

[44] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang, *Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices*, Black Hat (2017).

[45] Omid Mirzaei, Guillermo Suarez-Tangil, Jose M de Fuentes, Juan Tapiador, and Gianluca Stringhini, *Andrensemble: Leveraging api ensembles to characterize android malware families*, Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, 2019, pp. 307–314.

[46] Andreas Moser, Christopher Kruegel, and Engin Kirda, *Limits of static analysis for malware detection*, Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), IEEE, 2007, pp. 421–430.

[47] Hamad Naeem, Bing Guo, and Muhammad Rashid Naeem, *A lightweight malware static visual analysis for iot infrastructure*, 2018 International Conference on Artificial Intelligence and Big Data (ICAIBD), IEEE, 2018, pp. 240–244.

[48] Huy-Trung Nguyen and Quoc-Dung Ngo, *A novel graph-based approach for iot botnet detection*, International Journal of Information Security (2019), 1–11.

[49] Huy-Trung Nguyen, Quoc-Dung Ngo, and Van-Hoang Le, *Iot botnet detection approach based on psi graph and dgcnn classifier*, 2018 IEEE International Conference on Information Communication and Signal Processing (ICICSP), IEEE, 2018, pp. 118–122.

[50] Huy-Trung Nguyen, Quoc-Dung Ngo, Doan-Hieu Nguyen, and Van-Hoang Le, *Psi-rooted subgraph: A novel feature for iot botnet detection using classifier algorithms*, ICT Express (2020).

[51] Huy-Trung Nguyen, Doan-Hieu Nguyen, Quoc-Dung Ngo, Vu-Hai Tran, and Van-Hoang Le, *Towards a rooted subgraph classifier for iot botnet detection*, Proceedings of the 2019 7th International Conference on Computer and Communications Management, 2019, pp. 247–251.

[52] Michel Oosterhof, *Cowrie ssh/telnet honeypot*, 2016, Available at `https://github.com/cowrie/cowrie` [Online; Accessed Nov. 2020].

[53] Mete Ozay, Inaki Esnaola, Fatos Tunay Yarman Vural, Sanjeev R Kulkarni, and H Vincent Poor, *Machine learning methods for attack detection in the smart grid*, IEEE transactions on neural networks and learning systems **27** (2016), no. 8, 1773–1786.

[54] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow, *Iotpot: analysing the rise of iot compromises*, 9th {USENIX} Workshop on Offensive Technologies ({WOOT} 15), 2015.

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

[56] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow, *Iotpot: A novel honeypot for revealing current iot threats*, Journal of Information Processing **24** (2016), no. 3, 522–533.

[57] Marc-Oliver Pahl, François-Xavier Aubet, and Stefan Liebald, *Graph-based iot microservice security*, NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2018, pp. 1–3.

[58] Abdurrahman Pektaş and Tankut Acarman, *Classification of malware families based on runtime behaviors*, Journal of information security and applications **37** (2017), 91–100.

[59] Neha Runwal, Richard M Low, and Mark Stamp, *Opcode graph similarity and metamorphic detection*, Journal in Computer Virology **8** (2012), no. 1-2, 37–52.

[60] Hex-Rays SA., *Ida pro*, Available at `https://www.hex-rays.com` [Online; Accessed Nov. 2020].

[61] Igor Santos, Felix Brezo, Javier Nieves, Yoseba K Penya, Borja Sanz, Carlos Laorden, and Pablo G Bringas, *Idea: Opcode-sequence-based malware detection*, International Symposium on Engineering Secure Software and Systems, Springer, 2010, pp. 35–43.

[62] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A Selcuk Uluagac, *A survey on sensor-based threats to internet-of-things (iot) devices and applications*, arXiv preprint arXiv:1802.02041 (2018).

[63] Michael Sikorski and Andrew Honig, *Practical malware analysis: the hands-on guide to dissecting malicious software*, no starch press, 2012.

[64] Arunan Sivanathan, Daniel Sherratt, Hassan Habibi Gharakheili, Vijay Sivaraman, and Arun Vishwanath, *Low-cost flow-based security solutions for smart-home iot devices*, 2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), IEEE, 2016, pp. 1–6.

[65] Silvia Wahballa Soliman, Mohammed Ali Sobh, and Ayman M Bahaa-Eldin, *Taxonomy of malware analysis in the iot*, 2017 12th International Conference on Computer Engineering and Systems (ICCES), IEEE, 2017, pp. 519–529.

[66] Alireza Souri and Rahil Hosseini, *A state-of-the-art survey of malware detection approaches using data mining techniques*, Human-centric Computing and Information Sciences **8** (2018), no. 1, 3.

[67] Jiawei Su, Vargas Danilo Vasconcellos, Sanjiva Prasad, Sgandurra Daniele, Yaokai Feng, and Kouichi Sakurai, *Lightweight classification of iot malware based on image recognition*, 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), vol. 2, IEEE, 2018, pp. 664–669.

[68] Wen-Tsai Sung and Yen-Chun Chiang, *Improved particle swarm optimization algorithm for android medical care iot using modified parameters*, Journal of medical systems **36** (2012), no. 6, 3755–3763.

[69] Hiroaki Tanaka and Shingo Yamaguchi, *On modeling and simulation of the behavior of iot malwares mirai and hajime*, 2017 IEEE International Symposium on Consumer Electronics (ISCE), IEEE, 2017, pp. 56–60.

[70] Hüseyin Tirli, Abdurrahman Pektas, Yliès Falcone, and Nadia Erdogan, *Virmon: a virtualization-based automated dynamic malware analysis system*, International Information Security & Cryptology Conference, 2013.

[71] Wikipedia, *Remaiten*, 2018, Available at `hhttps://en.wikipedia.org/wiki/Remaiten` [Online; Accessed Nov. 2020].

[72] Liang Xiao, Xiaoyue Wan, Xiaozhen Lu, Yanyong Zhang, and Di Wu, *Iot security techniques based on machine learning*, arXiv preprint arXiv:1801.06275 (2018).

[73] Jaehak Yu, Hansung Lee, Myung-Sup Kim, and Daihee Park, *Traffic flooding attack detection with snmp mib using svm*, Computer Communications **31** (2008), no. 17, 4212–4219.

Vita

Candidate's full name: Nastaran Mahmoudyar

University attended:

Master of Computer Science
University of New Brunswick
2018-2021

Bachelor of Science in Computer Engineering
Iran University of Science and Technology
2007-2012

Publications:

None

Conference Presentations:

Hashemian, M., Moradi, H., Mirian, M.S., Tehrani-doost, M. and Mahmoudyar, N., 2016. *Recognizing mood using facial emotional features.* IEEE Transaction on Affective Computing