# Federated Learning Assisted IoT Malware Detection Using Static Analysis

by

Madumitha Venkatasubramanian

Bachelor of Computer Science and Engineering, SRM Institute of Science and Technology, 2019

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):** | Arash Habibi Lashkari,PhD, Faculty of Computer Science |
| | Saqib Hakak, PhD, Faculty of Computer Science |
| **Examining Board:** | Rongxing Lu, PhD, Faculty of Computer Science, Chair |
| | Haruna Isah, PhD, Faculty of Computer Science |
| | Mohsen Mohammadi, PhD, Faculty of Engineering |

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

December, 2022

# Abstract

The Internet of Things (IoT) has millions of connected devices; however, with the increased use of IoT devices, the threat of malware has increased rapidly. With technological advances, recent works have attempted to use Machine Learning and Deep Learning for IoT malware detection, but most of these techniques are centralized learning techniques, and they do not efficiently protect confidential user data. To address these issues in this research, we propose a Federated Learning-based approach that employs a Random Forest model for detecting IoT malware samples. Through the Federated Learning solution, we ensured that the local IoT device data remains locally and is not moved off the device. The results from our proposed model achieve an accuracy of 95% and efficiently classify the malware and benign samples. The overall comparative results between our proposed decentralized model and a centralized format demonstrate a significant improvement in the accuracy of malware detection while preserving the security of user data.

# Dedication

I dedicate my thesis to my mother, father, brother, family, friends and teachers for their endless support and encouragement from thousands of miles away. A special feeling of gratitude to my teachers, S. Niveditha, P. Amudha, M. Indumathy, Anu and Dr. S. Muruganandam who have supported and helped me achieve my dreams. I would also like to dedicate my thesis to Austin, Katherina, Rick and Midnight. This thesis is also dedicated to the people of Fredericton for showering me with their endless love and affection.

# Acknowledgements

I sincerely thank my professors, Dr. Arash Habibi Lashkari and Dr. Saqib Hakak, for their continuous support and guidance. They have motivated me and encouraged me throughout the process.

I would like to thank the University of New Brunswick, Faculty of Computer Science, for the opportunity and support and all my Professors who taught me during the program.

I thank Hillary Nguyen and Jason MacFarlane, International Student Advisors, for their immense support and encouragement during my travel to Canada. I also thank my co-op coordinators, Patrica Meng and Shelly Zimmerman, for their support and guidance during my co-op process.

I would also like to thank all my colleagues at 3D Planeta for being very kind and encouraging during my internship period.

I would like to thank my examining committee Dr. Rongxing Lu, Dr. Harunah Isah and Dr. Mohsen Mohammadi for their valuable feedbacks.

# Table of Contents

# List of Tables

# List of Figures

# Abbreviations

| | |
|---|---|
| *IoT* | Internet of Things |
| *CFG* | Control Flow Graph |
| *PSI* | Printable String Information |
| *FCG* | Function Call Graph |
| *ELF* | Executable and Linkable Format |
| *DDoS* | Distributed Denial-of-Service |
| *ML* | Machine Learning |
| *DL* | Deep Learning |
| *RNN* | Recurrent Neural Network |
| *DGCNN* | Deep Graph Convolutional Neural Network |
| *LR* | Logistic Regression |
| *SVM* | Support Vector Machine |
| *KNN* | k-Nearest Neighbours |
| *DT* | Decision Tree |
| *FL* | Federated Learning |
| *RF* | Random Forest |

# Chapter 1

# Introduction

## 1.1 Introduction

The Internet of Things (IoT) is an emerging technology rapidly evolving communication. IoT permits exchanging of meaningful information and knowledge across IoT devices and systems to create value for humans [16]. IoT involves billions of connected devices, generating vast amounts of data [36]. These devices may include sensors, smartphones, computers, or home appliances. These devices are connected to the internet and each other through heterogeneous access networks [36]. It can be described as connecting devices or things across the internet to send or receive data [42]. IoT devices are used in several industries and have proven helpful for remote health monitoring, early diagnosis, and elderly care for the healthcare sector [18] and reducing the necessity to meet with doctors in person [68]. The research on agriculture using IoT has also risen in the last two decades around the crop, soil, and microclimate monitoring [8]. In addition, several industries utilize IoT devices, including the finance sector [17], retail [15], vehicle monitoring systems [78], and the manufacturing industry [20].

With the increase in the application of IoT devices across several industries, the increase in threats caused by malware has also increased rapidly. The increased number of existing and new malware variants has made protecting IoT devices and networks challenging. The malware can hide in the systems and disable its activity when there are attempts to discover and detect them. Malware is a harmful code that steals sensitive information, damages the devices, disrupts the network and consumes memory [10]. Several types of malware include viruses, trojan horse, worms, spyware and backdoor. With the increase in security threats, improving the security of IoT devices and protecting them against malware is critical.

## 1.2 Types of Malware

Malware are malicious software that tries to infiltrate computer systems and they try to damage them without the user's consent. Different types of malware exploit the vulnerability in IoT devices. These malware can be categorized based on their functionality features. The different types of malware are discussed below, along with their functionalities.

1. Virus

   A virus [64] is a malware that runs on the user's computer without their knowledge and infects it when the code is accessed and executed. The viruses are capable of copying themselves and replicating inside the system. In addition, it can steal the user's sensitive information and damage the host system with the help of a malicious program.

2. Ransomware

   Ransomware restricts access to the computer system and demands a ransom amount to be paid to regain access to the system. It propagates as a trojan or a worm, encrypts the user's file, and makes it inaccessible. The ransomware is mostly spread through phishing emails.

3. Trojan Horse

   The Trojan Horse hides as a general program and tricks the user into installing them into the system. It allows the hacker to gain remote access and higher system administration privileges.

4. Worms

   Worms deploy over a network by exploiting the operating system's vulnerabilities and running the same application through the internet. As a result, worms do not have to be accessed or executed and damage the host networks through web server overload.

5. Fileless virus

   A fileless virus is a malware that does not reside on the hardware of a computer and works in memory-based environments. It gains remote access to the system.

6. Botnets

   A botnet is a group of devices connected to the internet, each running a bot. An attacker can use botnets to perform Distributed Denial-of-Service (DDoS) attacks, steal data, send spam, and gain access to a device. In addition, the attacker can control the botnet using command and control (C&C).

7. Spyware

   Spyware is a malicious software that gathers data from the user without their consent and sends it to third parties. The activities of spyware include monitoring, collecting keystrokes, and harvesting data such as credit card numbers, financial data, and account credentials [64].

## 1.3 Conducive environment for the execution of IoT malware

IoT devices are prone to attacks, including physical attacks, network-layer attacks, and application-layer attacks [65]. The attacker exploits the vulnerabilities present in the targeted system. There are several reasons for the execution of malware such as outdated firmware, weak authentication, connectivity, and resource-constrained devices. We will be discussing the outlined reasons in detail.

- Outdated Firmware - Firmware updates the functionality and features of a device. Usually, outdated firmware does not have security patches if new vulnerabilities are found [67]. Therefore, the attackers can exploit this vulnerability and gain access to the rest of the system.

- Weak Authentication - IoT devices usually have an easy installation procedure for the people to use the devices. Most users either reuse their credentials or do not change the default credentials, which becomes an easy target for attackers [23].

- Connectivity - Many IoT devices are connected to the internet almost always. This creates open ports which attract attackers easily. In addition, most IoT devices are resource-constraint and have fewer security policies.

- Resource-constrained - Most IoT devices, such as smart watches, CCTV cameras, and bluetooth-operated devices, are resource constrained and heterogeneous, making it easier for attackers to target the system [34].

Today there are various industries reliant on IoT devices, and there is an increase in security issues faced by the industries. With the design complexity and lack of security due to outdated firmware and weak authentication, IoT devices are targeted by cybercriminals, and malware compromises IoT devices [7]. Therefore, it is critical to improve the security of IoT devices and protect them against malware. Many types of research and studies are conducted to protect IoT devices against malware, and a few such methods involve using Machine Learning [16],[72], and Deep Learning [69],[39]. However, most of them are centralized learning techniques. As a result, they do not efficiently protect confidential user data as they share the user data with a centralized model and receive back the updated model from the centralized servers causing several security concerns. We propose a Federated Learning-based approach to address these issues in this research. The next section will give an overview of the Federated Learning technique.

## 1.4 Definition of Federated Learning

Federated Learning (FL) is a new branch of AI where the Machine Learning (ML) models are trained locally on the devices such as mobile phones and other smart devices [14]. The devices present in the FL setup do not exchange their local data but instead shares the parameters and gradients of the local model with a global model maintaining the security of the user. The global model resides on a server and aggregates all the model updates obtained from the local models by averaging the parameters of each individual model. By this method, each individual model learns collaboratively from a global model.

## 1.5 Problem Statement

As discussed in the previous sections, with the increasing number of applications of IoT devices across several industries, it is crucial to protect these devices from malware. Previous studies used Machine Learning and Deep Learning techniques to detect and classify malware samples. However, most of these are centralized learning techniques. Centralized learning techniques send confidential user data to a centralized server where they apply Machine Learning or Deep Learning techniques to detect and classify malware and benign samples. Therefore, sharing sensitive user data affects user security and does not efficiently protect confidential user data. In this study, we address this issue using a decentralized learning technique called Federated Learning (FL). In FL, confidential user data are not moved off the device and remain locally.

The main idea behind FL is to train the user data on-device without sharing its private data to a central server, as present in centralized learning techniques. In FL, the devices in the FL setup share their model parameters instead of their confidential user data. This way, the model parameters are transferred to and from the FL server rather than the data itself. The global model in the FL setup aggregates the model updates and parameters obtained from each local model in the devices connected to the network. The aggregated model parameters are then shared with the local models present at each device for the next round of the FL process. This protects user data and provides a secure system.

## 1.6 Summary of Contributions

The main purpose of this research is to protect the security of confidential user data by using a Federated Learning technique without moving off the local data from the device, detecting malware and benign samples. In summary, the following are the contributions of this thesis:

1. We are proposing to integrate Federated Learning and IoT for IoT malware detection.

2. Feature extraction using Graph-based and Non-Graph-Based techniques using static analysis.

3. We propose a two-stage architecture including a global model with ideal parameters and an architecture for IoT malware detection using Federated Learning.

## 1.7 Thesis Organization

The rest of the thesis is organized as follows:

1. Chapter 2: *Background* discusses the required definitions and background for understanding the previous and current studies, including the nature of IoT malware and the types of malware analysis.

2. Chapter 3. *Literature Review* discusses the previous studies on static analysis using Deep Learning and Machine Learning techniques discussing in detail the features and algorithms used. This chapter also highlights the advantages of integrating Federated Learning and IoT along with the detailed workflow and topology of Federated Learning.

3. Chapter 4: *Proposed Method* discusses the motivation of the proposed method along with a detailed discussion on the two-stage architecture and explains each of the main components.

4. Chapter 5: *Experiments & Results* represents the results of the research implementation, and in this chapter, we discuss the dataset, the experimental setup, features, metrics and results of the proposed approach.

5. Chapter 6: *Conclusion & Future Work* concludes the thesis with the contribution summary, challenges and some remarks on the future work.

# Chapter 2

# Background

## 2.1 Overview

In this chapter, we provide the necessary background to understand the current studies in this field. We have discussions on the nature of IoT malware and the different types of malware analysis.

In section 2.2, we discuss the various components involved in attacking vulnerable devices. Later in section 2.3, we discuss the types of IoT malware analysis, including static, dynamic and hybrid analysis, and provide a taxonomy illustrating the different types of features used.

## 2.2 Nature of IoT Malware

Most malware families are designed to target personal computers running on Microsoft Windows, the most popular operating system. IoT devices are built upon different CPU architectures and have become an attractive target for attackers. The IoT malware performs brute-force attacks to gain access to the devices. The Linux.Hydra was the first DDoS-capable IoT malware that appeared in year 2008 [7]. The IoT malware developers developed several variants of Linux.Hydra, including Psybot, Chuck Norris, and Tsunami, emerged in the upcoming years. The Tsunami is the ancestor of Bashlite, and from Bashlite, the Mirai malware inherited and evolved more and more complex in 2016 [7]. In Figure 2.1, we can see the evolution of IoT malware over the years.

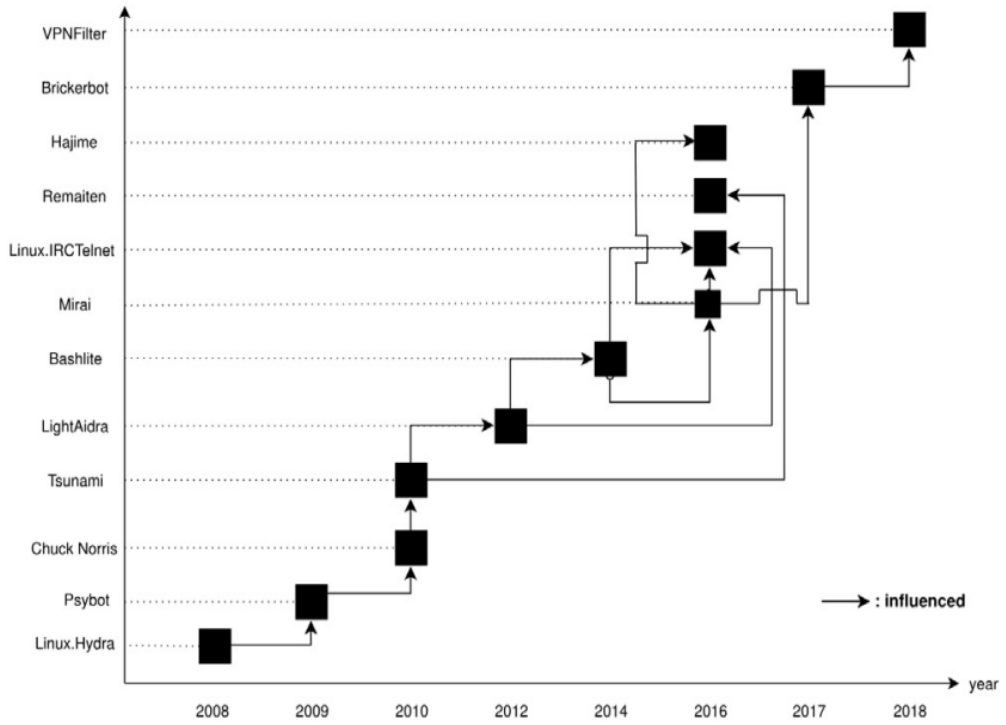


Figure 2.1: Evolution of IoT malware [7]

The different types of malware have different modes and natures of exploiting the vulnerabilities of the targeted system. For example, the malware can exploit network-based vulnerabilities or use operational business functionality through available network shares [1]. Incident response and malware eradication efforts are challenging when the malware propagates utilizing the infrastructure. Earlier in the section, we discussed the evolution of certain malware. To know more about the nature of the malware, we will further discuss the propagation methods with an example of one of the most popular IoT botnets, Mirai.

The Mirai botnet consists of five major components [43], and all of these work independently to compromise vulnerable devices and launch massive DDoS attacks. In addition, all these components are distributed geographically, which makes them difficult to track [43]. The following are each of the components explained in detail, along with their functionalities.

1. Bot - A bot [72] is a malicious component in the network; a bot could be any IoT device connected to the network. It is a slave node and acts on the attackers' behalf, taking instructions from the attackers and executing them in the network. Each bot scans for the nearby vulnerable device and reports it to the report server. The bots attempt brute-force attacks using default usernames and passwords.

2. Command and Control Server (C&C) - In the C&C server, the attacker is the one who controls the botnets and sends out instructions that are carried out by the botnets.

3. Report Server - The report server contains information about the vulnerable nodes and their stolen login credentials. The bot communicates this information to the report server when it locates a vulnerable node.

4. Loader - The loader obtains information from the report server and exploits these vulnerabilities to change the node into a bot.

5. Webserver - It hosts the precompiled bot binaries for multiple architectures. The loader identifies the appropriate architecture and downloads the corresponding binary from the web server [43].

## 2.3 Types of IoT Malware Analysis

Malware Analysis studies a malware sample's impact, functionalities, origin, and potential. It helps understand the behaviour and purpose of a suspicious file, reduces the false positives, and helps to determine how extensive is a malware incident. There are three types of malware analysis: dynamic [49], static [46], and hybrid [2]. Each of the techniques has its strengths and weakness compared to the others. The dynamic analysis uses a behaviour-based approach. Compared to static analysis, dynamic analysis is effective against all types of malware.

Static analysis is ineffective against sophisticated malware but, compared to dynamic analysis, is cost and time efficient. The static analysis involves file fingerprinting and virus scanning, and it searches the body of the malware for strings [54]. The limitations of static and dynamic analysis inspired researchers to develop a hybrid analysis that involves the benefits of both static and dynamic analysis [45]. This section discusses each technique in detail and provides a taxonomy and features.

### 2.3.1 Static Analysis

The static approach analyzes and detects malicious files without executing them [46]. In static analysis, the analysts reverse the executable files into assembly code to better understand the malware activities. The significant advantage of static analysis is that it can observe the malware's structure and scalability. Observing the malware's structure helps explore all the possible execution paths in the malware sample and makes the static approach effective in solving heterogeneous issues in IoT devices [46].

The static analysis relies on extracting certain characteristics: Control Flow Graph (CFG), Function Call Graph (FCG), opcodes, strings, and file headers. Then, the assembly code is disassembled [33] using tools like Radare2 [73] and IDA Pro [40].

These characteristic features can be categorized as graph-based features and non-graph-based features. The two types of features are discussed in detail below:

- *Graph-based features:* The Control Flow Graph (CFG) is the most popular feature in graph-based features. CFG is a directed graph representing all the possible execution paths in a program where each vertex represents a basic block, and each directed edge is the control flow between the blocks [7]. The experimental results in [27] have shown that the IoT malware contains fewer nodes and edges than android malware. The authors of the paper [28] build a detection mechanism for IoT malware using CFGs. The paper shows that the IoT malware has a more significant number of edges despite the smaller number of nodes.

In [47], the authors propose preserving the malware's integrity by extracting the CFG of malware as feature information. Here, a packed malware's control flow graph consists of unpacked and local CFG. The paper [62] proposes a new algorithm in the CFG feature based on dynamic programming to detect the malware with fast processing time efficiently.

The next type of graph-based feature is the Function Call Graph (FCG). The FCG is also a directed graph constructed from programs where the vertices specify the functions and the edges define the caller-callee relationship between functions [81]. Figure 2.2 shows an example of FCG for a Linux Mirai sample [58].

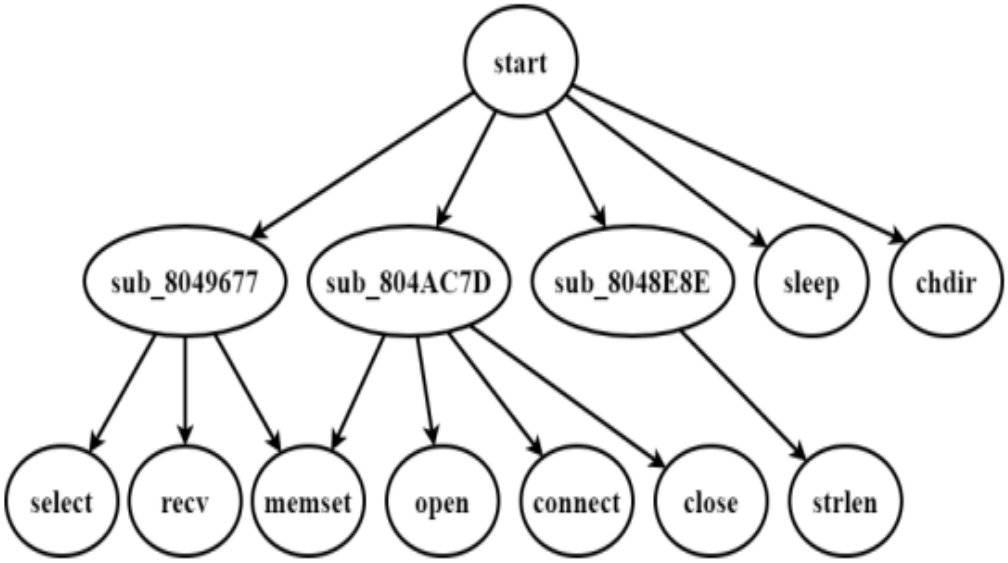


Figure 2.2: Part of Function Call Graph for Linux.Mirai sample [58]

In this study [7], there is another type of graph-based feature: the opcode sequence graph. The opcode sequence graph represents an executable file as the opcodes have a suitable structure to be represented as a graph [44]. Representing an executable file as a graph allows graph-based implementation methods

like graph compression and embedding to distinguish between malware and benign files.

- *Non-graph-based features:* There are several non-graph-based static features, such as opcodes, ELF headers, and strings. One of the functionalities most IoT malware supports is the Command & Control server connection [9]. Therefore, the C&C server and IP address might be available in printable strings of IoT malware binary. In the paper [30], the authors have obtained the statistical, structural, and string features. The statistical features have been obtained using course-grained clustering, the structural features are obtained using fine-grained clustering, and the string features are obtained using N-grams.

  In one of the survey papers [7], the authors mentioned that the string features that include information such as the IP addresses and URL connect take the least time for feature extraction. For the opcodes, the malware file is decompiled to extract opcodes and utilized for malware classification [9]. The authors in [3] extracted opcode features for malware and benignware using the objdump tool. In the paper [5], the authors have extracted the opcode sequences using fuzzy and fast fuzzy pattern trees.

### 2.3.2 Dynamic Analysis

The dynamic approach monitors the executables during the run-time period and detects abnormal behaviour. It observes behaviour information such as network activities, system calls, file operations, and registry modification records [49]. The dynamic analysis monitors the execution process and is resource-intensive, time-consuming, and expensive for constructing a virtual environment. In some cases, the malware could infect real environments. Although they are resource intensive, the

dynamic analysis is effective against all types of malware.

The main advantage is that it analyses the run-time behaviour of a program which is hard to obfuscate [2]. Some examples of the features in the dynamic analysis include memory, network, system call sequences, process ID, and parent process ID [49]. In [37], the authors design and implement an automatic, virtual machine-based profiling system to collect IoT malware behaviour, such as API calls and system calls. The method converts multiple sequential data into a family behaviour graph for analysis.

The paper [66] proposes a dynamic analysis methodology by preparing an analysis tool and running the malicious samples in a controlled environment to investigate them. Meanwhile, the authors of [60] propose a method for malware classification based on analyzing the sequences of system calls and using an attention-based LSTM model for malware classification. Finally, in [75], the paper discusses the techniques performed by malware to evade detection in a dynamic analysis environment.

### 2.3.3 Hybrid Analysis

The hybrid malware analysis integrates both static and dynamic features. In the paper [2], the authors have combined the static and dynamic features to utilize the benefits of both techniques. It utilizes the string features for the static analysis and API call sequence extraction for the dynamic analysis. The combination of both features improved detection accuracy compared to the stand-alone techniques. In the paper [33], the authors have used hybrid analysis using an entropy-based method to differentiate packed malware samples from non-packed ones.

The authors of [32] use two-stage hybrid malware detection to protect IoT devices from obfuscated malware. The method consists of two stages where after extracting opcode features using static analysis, the benign files are detected using a bidirectional long short-term memory model. In [48], the authors propose to use bidirectional long short-memory (Bi-LSTM) along with a spatial pyramid pooling network (SPP-Net) for smart IoT.

The advantage of hybrid analysis is that certain actions hidden in the run-time might be detected while unpacking the binary files or viewing them as assembly code. The detailed taxonomy of the IoT malware analysis can be seen in Figure 2.3.

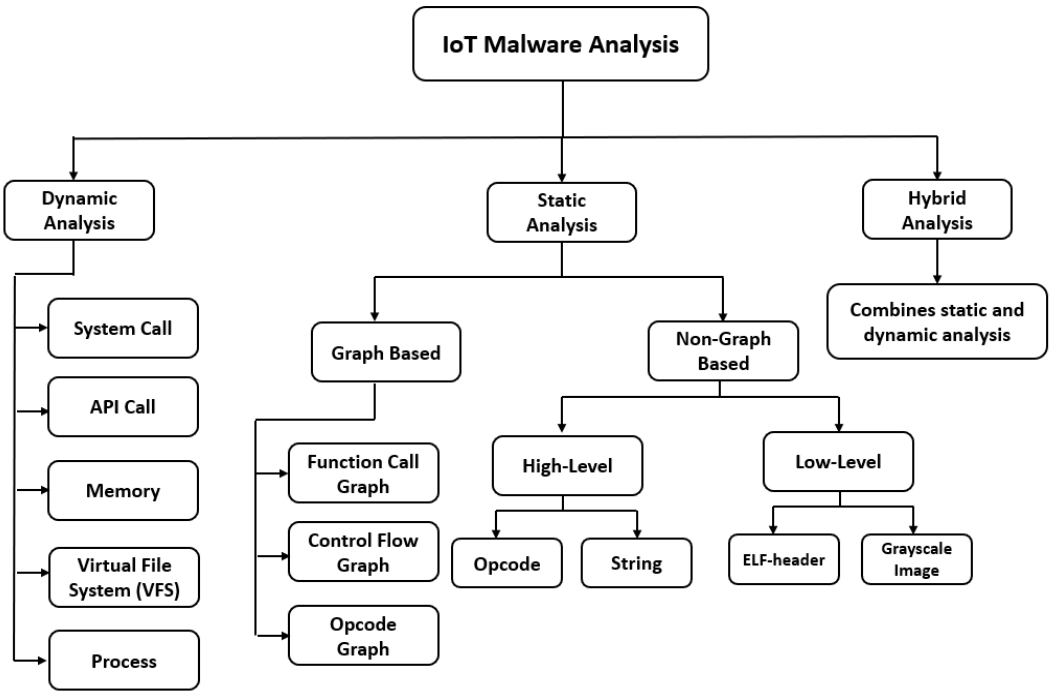


Figure 2.3: Taxonomy of IoT malware Analysis

## 2.4 Concluding Remarks

This chapter provided an overview of the necessary background information for the thesis. First, we discuss in detail about IoT malware providing a detailed explanation of the nature of IoT malware and the various components involved in them. We also broadly discuss the various types of malware analysis, providing a taxonomy of the different features used for each method. After understanding the background and definitions in this chapter, in Chapter 3, we will discuss the related work done in IoT malware analysis, focusing on static analysis.

# Chapter 3

# Literature Review

## 3.1 Overview

The rapid growth of IoT across several industries has increased security concerns. This chapter first reviews the approaches used for IoT malware detection using static analysis. Then, based on the technique used, we group the current studies based on two main categories, Graph-Based and Non-Graph-Based methods. The following discusses each of the categories along with the techniques used.

1. Graph-Based Related Work
    - Machine Learning
    - Deep Learning
2. Non-Graph-Based Related Work
    - Machine Learning
    - Deep Learning

This chapter presents a comprehensive survey of the current centralized learning techniques used and the significant drawbacks of not protecting the sensitive user data as they are shared to a central server for training the models. At the end of the chapter, we discuss the advantages of integrating Federated Learning with IoT and the workflow of Federated Learning.

## 3.2 Literature Review

In this section, we gather the existing studies on IoT malware detection using static analysis. We will discuss the graph-based and non-graph-based approaches using different techniques to detect and classify IoT malware samples. First, we discuss the graph-based approach. Static analysis using the graph-based approach can help generalize and represent the structured and complex information about the behaviour of the malware. However, the graph-based approach takes a long time for feature extraction.

Here in this section, we discuss the graph-based approach based on the two methods, Machine Learning and Deep Learning, used as centralized learning techniques. In this part, we present a detailed study of all the recent and current research works related to IoT malware detection and classification.

Next, we focus on a non-graph-based approach. The non-graph-based approach takes much less time for the feature extraction but is relatively weak as the malware can use obfuscation techniques. In the non-graph-based approach, we focus on the recent research on Machine Learning, Deep Learning and other techniques used to detect and classify IoT malware. Finally, we review all the research related to IoT malware detection analysis to get a more accurate view. This literature review will

help researchers to analyze the existing gaps.

### 3.2.1 Graph-Based Related Work

In this part, we examine the existing studies using graph-based techniques in static analysis. We discuss in detail the Machine learning and Deep Learning based techniques used in static analysis for IoT malware detection and classification.

#### 3.2.1.1 Machine Learning

Several recent research works focus on using Machine Learning techniques for IoT malware detection and classification using graph-based features. In the paper [6], the authors propose PSI-rooted (Printable String Information) subgraph-based features for detecting IoT malware. The authors propose to use a limited number of features with precise behavioural descriptions to reduce the time and space required for processing. The proposed method is evaluated using five Machine Learning classifiers: Random Forest, Decision Tree, Bagging, k-Nearest Neighbor, and Support Vector Machine (SVM). The Random Forest classifier performed better than the others.

In paper [79], the authors used Function Call Graph (FCG) to extract the features, and the proposed method collected samples from seven different CPU architectures. The authors used SVM and Multilayer Perceptron(MLP), and MLP performed better in accuracy. In [81], the authors propose using a Function Call Graph Vectorization (FCGV) and statistical features. The integrated vector obtained from both features is fetched into a Random Forest model. Figure 3.1 represents the overview of the FCGV model. In [62], the authors propose to use Control Flow Graph (CFG) based features for IoT malware detection. The proposed method achieves faster processing time and uses less memory. In [29], the authors have also used CFG to highlight the difference and similarities between IoT malware samples and Android samples.

The proposed method extracts CFGs as abstract representation to characterize and highlight the difference in graph representation. The proposed method is evaluated using different machine learning classifiers.

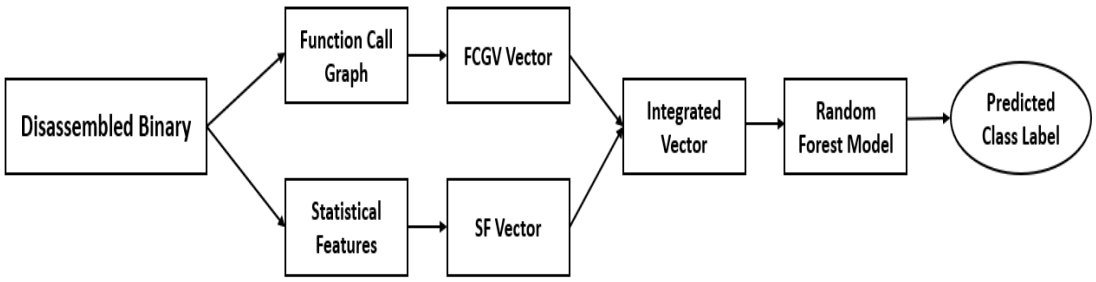


Figure 3.1: FCGV Architecture [81]

The authors in [27] use Control Flow Graphs (CFG) to analyze the general characteristics of IoT malware. In [44], the authors utilize opcode sequences for malware detection. The proposed method uses opcode graphs of every executable for its feature extraction. The authors compare the proposed method against different Machine Learning classifiers, and Random Forest performed better than the others.

In [63], the authors extract CFG features based on the opcode. The paper proposes CFDVex, which extracts CFG features based on Vex. The idea of the CFD method is that it calculates the n-gram of the opcode stream from all execution paths of CFG. Whereas CFDVex extracts CFG features based on Vex instead of opcode, it calculates n-gram of Vex concatenated from all the execution paths of CFG. The proposed method uses an SVM classifier and compares the performances of different levels of the CFDVex. In [56], the proposed approach extracts PSI graphs from executable files and extracts two levels of features such as graph-level and subgraph-level. The proposed method compares the performances of different machine learning classifiers, and the Decision Tree performed the best among the others.

#### 3.2.1.2 Deep Learning

Several recent research works focus on using Deep Learning techniques for IoT malware detection and classification. In [28], the authors used Control Flow Graph (CFG) to detect IoT malware. The proposed method analyses android malware and IoT malware and shows that IoT malware samples have larger nodes showcasing higher complexity and flow structure. The proposed method uses Convolutional Neural Network (CNN) and highlights the differences between IoT and android malware. The authors in [57] proposed using Printable String Information (PSI) graphs. The proposed method extracts the vertices' local substructure features and defines a consistent ordering of the vertex. The method uses CNN, where the convolutional and dense layers read the sorted graph representations to make predictions.

In [58], the authors extracted high-level features from Function Call Graph (FCG) to detect IoT malware samples. The proposed method extracts PSI graphs from the FCG, where the PSI graph shows the structure of the relationship among functions containing printable strings in executable files. Figure 3.2 shows the PSI graph of Linux.Bashlite sample. The authors utilized CNN and compared the detection results between FCG and PSI graphs. In [47], the authors use CFGs of packed malware and analyze them dynamically to strip the unpacking routine from the graph. The proposed method used a Deep Graph Convolutional Neural Network (DGCNN) to classify the malware samples.

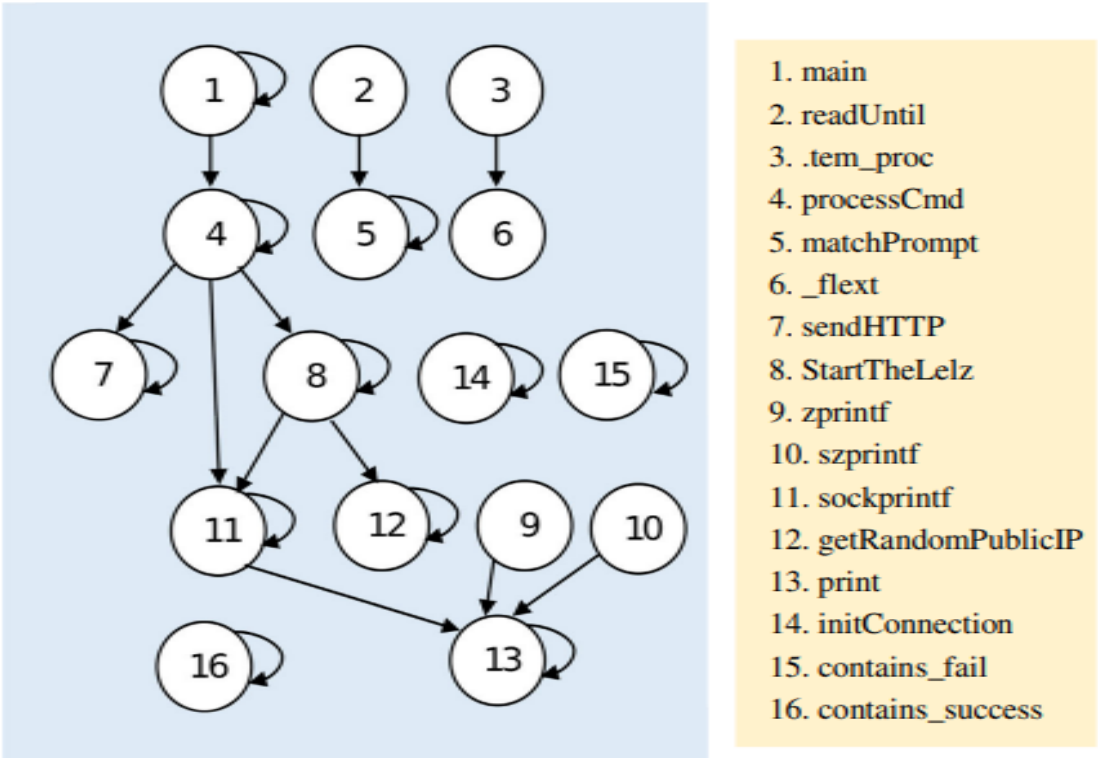


Figure 3.2: PSI Graph of Linux.Bashlite sample [58]

In [24], the authors propose to use a fine-grained hierarchical learning approach for IoT malware detection that uses CFG-based behavioural patterns for adversarial IoT malware detection. The proposed method uses Deep Learning techniques and compares the performances with Machine Learning techniques. In [80], the authors extract attributed CFGs from actual CFGs. The attributed CFGs contain structural information and the number of vertices. For malware classification, it aggregates these attributes over all the vertices in the attributed CFG in an organic manner, depending on its graph structure. This is achieved using DGCNN.

After that, the authors in [22] propose using an attributed Control Flow Graph (ACFG), combining ambient information with binary code information and then transforming the graph using a graph embedding algorithm. Finally, this is fetched into a Deep Neural Network (DNN) for IoT malware detection. Then, in [26], the

authors propose the Graph Embedding and Augmentation method to preserve the functionality and practicality of classifying IoT malware samples. The proposed method uses deep learning techniques and reduces the misclassification of benign and malicious samples. In [25], the authors propose a subgraph of the previous technique called Sub-graph Embegging and Augmentation (SGEA) to detect manipulated features generated by adversarial samples.

Other than Machine Learning and Deep Learning techniques, recent research has also focused on using different techniques, including reinforcement learning for IoT malware detection and classification. For example, in [55], the authors generate PSI walks from PSI graphs using reinforcement learning and feed this into a shallow LSTM network for IoT malware detection. The method uses a dataset of 6165 IoT botnet samples and 3845 benign samples.

### 3.2.2 Non-Graph-Based Related Work

In this part, we provide an overview of non-graph-based approaches. We will discuss in detail all the current and recent research work for IoT malware detection and classification. Furthermore, this section discusses all the different techniques used, including Machine Learning and Deep Learning, and the features used to detect and classify IoT malware.

#### 3.2.2.1 Machine Learning

In [30], the authors propose to use statistical and string features to develop IoT malware signatures. In addition, the authors use the features to cluster IoT malware families and to distinguish between malware and benign samples. The string features are extracted using N-gram text analysis, and the statistical features contain

code-level statistics. The proposed detection system contains two sections: offline signature generation and online detection. The first section has three clustering stages: coarse-grained, fine-grained, and cluster merging. The proposed method uses K-means clustering for coarse-grained clustering. Fine-grained clustering uses Bindiff for structural similarity. Finally, cluster merging merges the generated clusters that share high similarities. As a result, the proposed method has higher detection rates than previous Opcode N-gram techniques.

In [77], the authors used Levenshtein Distance (LD) as the similarity metric between ELF binaries. The LD calculates the difference between two input strings by counting the minimum number of operations required to transform one string to another. In the proposed method, LD was utilized to check the likelihood of two code segments being from the same source. The proposed method compares the performance of KNN and Support Vector Machine (SVM). In [51], the authors use printable strings to input Machine Learning models for IoT malware detection. The proposed method develops a malware analyzer that predicts malware on different CPU architectures based on the learnings from a single CPU architecture. Figures 3.3 and 3.4 show the difference in the Opcode of Mirai in different CPU architectures. The authors use three different Machine Learning classifiers: Random Forest, KNN and SVM. The performance of Random Forest is better in terms of accuracy than others.

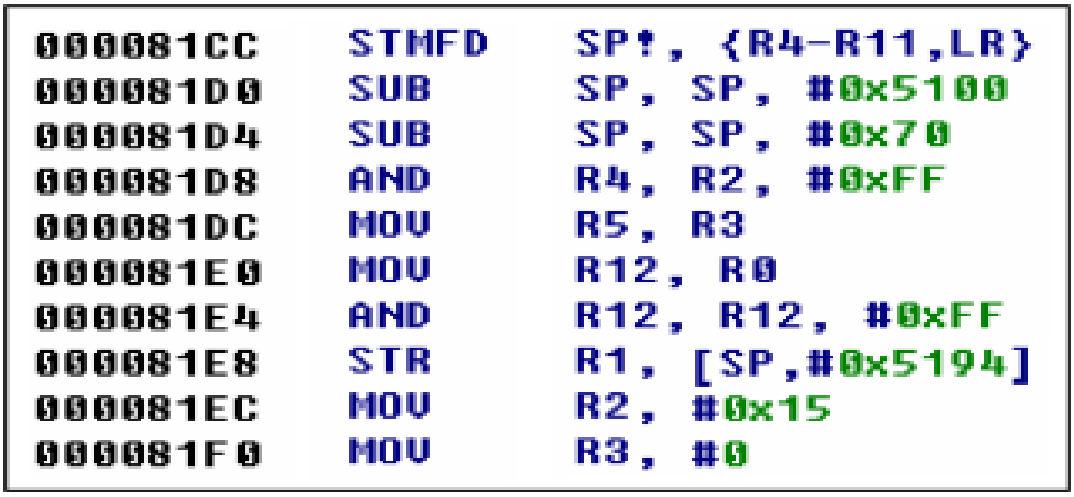


Figure 3.3: Opcode of Mirai (ARM) [51]

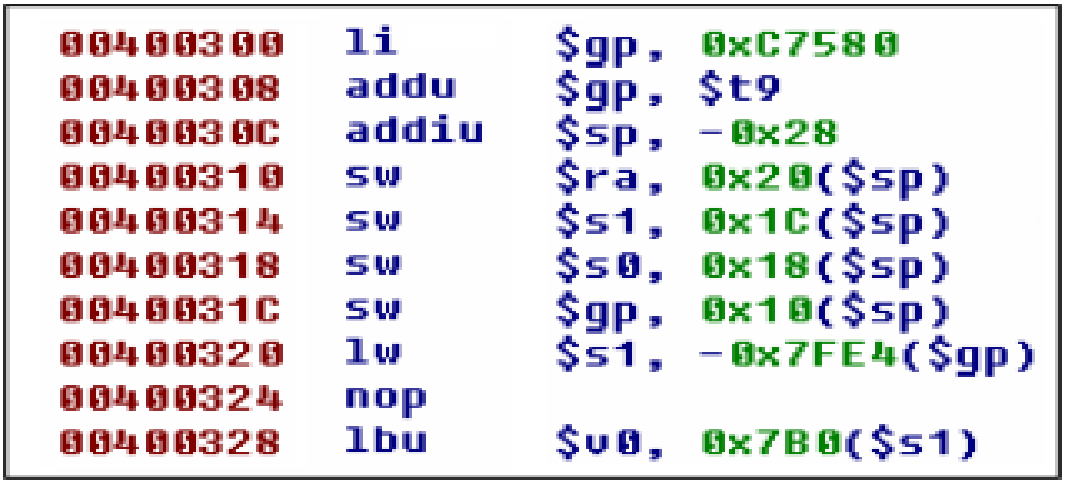


Figure 3.4: Opcode of Mirai (MIPS) [51]

The authors in [76] use the discriminating information stored in the byte sequence and leverage Machine Learning to detect and classify IoT malware. The input to the Machine learning models is the entry point of byte sequences. An executable file's entry point is where the first instructions of a program are executed [76]. For malware detection and classification, four different Machine Learning classifiers are used, including SVM, KNN, Naive Bayes (NB) and Multilayer Perceptron (MLP).

The performance of the SVM classifier was best in terms of the accuracy of detecting and classifying IoT malware and its family. In [46], authors propose using a new feature based on control statement shingling that requires smaller dimensions than opcode n-gram-based features. The proposed method compares the performance of Random Forest and Logistic Regression, and the Random Forest classifier works efficiently in terms of taking lesser training time.

In [50], the authors propose an early detection of IoT malware using Machine Learning (ML). The proposed method has five modules: ML classifier, ML model constructor, feature database, policy and sub-sampling module. The proposed method uses three different classifiers, NB, Random Forest and KNN and compares the performances based on accuracy, and KNN performed the best.

In [41], the authors propose extracting five features, including a novel Windows Entropy Map (WEM) image to represent malware with variable length and a set of frequently used APIs analyzed for shorter processing time. The proposed method tries to achieve fast low-dimensional feature preparation, low storage and faster learning. The effectiveness of the proposed approach is validated using tree ensemble techniques such as AdaBoost, XGBoost, Random Forest, extra trees and rotation trees. Compared to the five methods used, the XGBoost performance is better in terms of accuracy. In [38], the proposed method uses the detected maximal frequent patterns (MFP) of opcode sequences to classify malicious and benign IoT samples. The proposed method uses different machine learning classifiers, including KNN, SVM, MLP, Random Forest and Decision Tree. The performance of KNN and SVM were almost similar and performed better than the other classifiers. The proposed method also achieves high accuracy in detecting unseen malware samples.

In [74], the authors of the paper collected seven different static features based on the numbers and types of behaviours, the architecture targeted, shared library requirements in the system, the range in size for suspicious samples, and the ability to cloak characteristics through packing. The proposed framework can be seen in Figure 3.5. The method used an SVM classifier and achieved high accuracy in detecting IoT malware. In [53], the authors use static analysis and extract features, including file name, MD5 hashes, string features and opcode sequence. Furthermore, the authors used dissembler tools like IDA pro to reverse-compile the executables and describe the patterns of malicious content. Finally, they compared the performance of the proposed technique using the Decision Tree, Random Forest and Naive Bayes. The performance of the Random Forest model outperformed the others in accuracy.

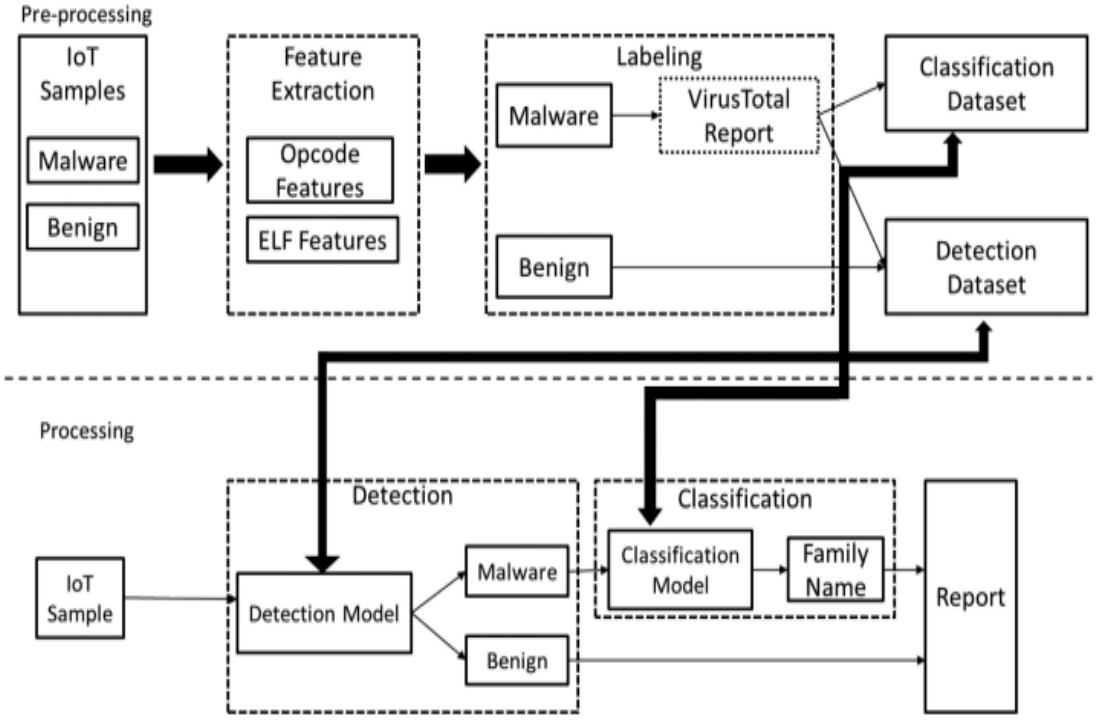


Figure 3.5: Overview of IoT malware framework for analysis [74]

In [16], the authors propose an automated machine learning (AutoML) perspective for IoT anomaly detection problems. The technique aims to automatically select, construct, tune, and update machine learning models to achieve the best performance on specified tasks. The proposed method uses the IoTID20 dataset for the proposed method. In [19], a Light Gradient Boosting Machine (LGBM) is proposed to identify malicious activities. The proposed method uses Genetic Algorithm (GA) for hyperparameter tuning, and the proposed approach achieves higher accuracies than previous approaches.

#### 3.2.2.2 Deep Learning

In [59], the authors have proposed using sequences, images and assembly for IoT malware detection. They compare the three techniques using Convolutional Neural Networks (CNN). The architecture for the assembly instructions using CNN can be seen in Figure 3.6. The proposed method shows that the sequences and assembly instructions perform well using CNN. In [3], the paper proposes to use Long Short-Term Memory (LSTM) for IoT malware based on Opcodes sequences. The proposed approach is evaluated for ARM-based IoT applications. The performance of the proposed method is compared with conventional Machine Learning models, and the proposed approach outperforms them. In Figure 3.7, the architecture of the LSTM structure for IoT malware detection is present.

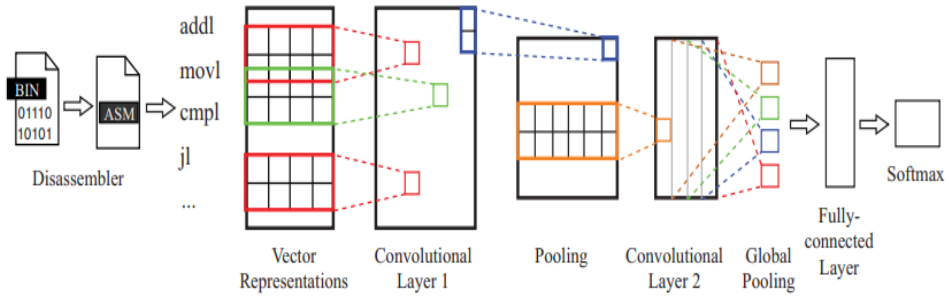


Figure 3.6: CNN Architecture on assembly instructions for IoT Malware Detection [59]

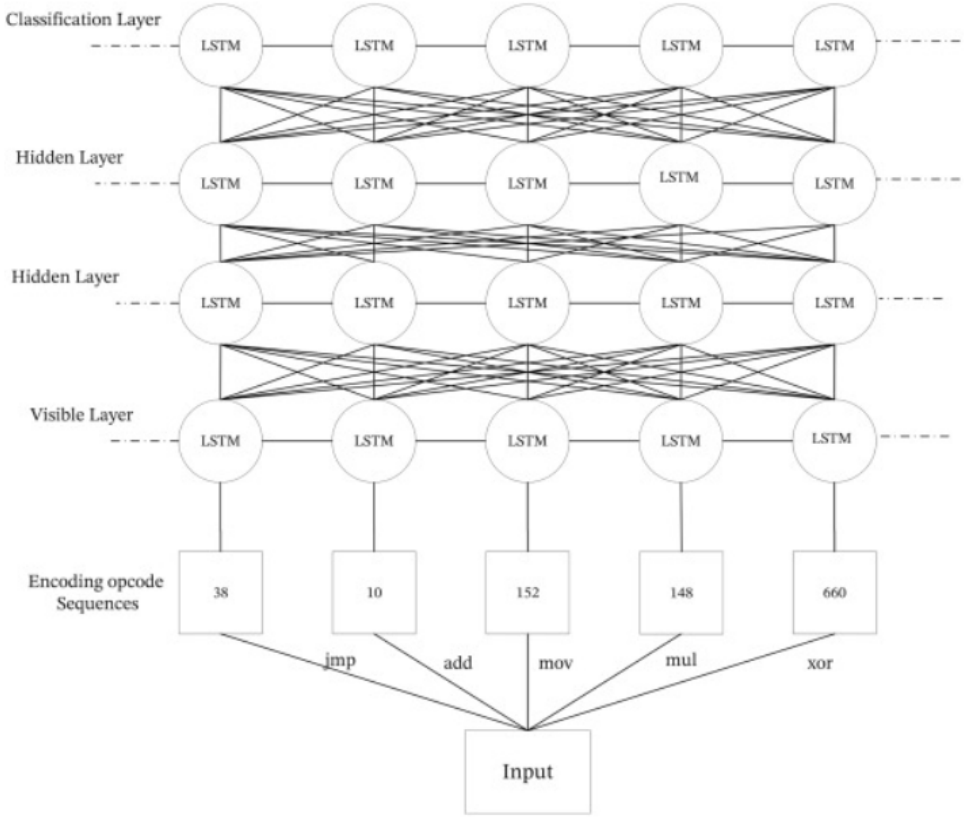


Figure 3.7: LSTM Structure for IoT Malware Detection [3]

In [4], static analysis utilizes the snapshot of the behavioural data to identify if it is malicious. The paper proposes to use an ensemble of Recurrent Neural Networks (RNN) to predict the executables. The proposed RNN model outperformed machine learning classifiers, such as Decision Tree, SVM, Naive Bayes and KNN. In [39], the authors propose to extract features from strings and image-based representations of the executable binaries. The system proposes a multilevel Deep Learning malware classification system using these extracted features for IoT malware detection. In [71], the authors have also used image-based techniques where they propose to use a lightweight approach for detecting IoT malware. The proposed method extracts malware images where a one-channel gray-scale image is converted from a malware binary. The method uses a lightweight Convolutional Neural Network for classifying their families. Table 3.1 shows the existing works on IoT malware detection and classification and the method and approaches used.

Recent research works have also used other techniques other than Machine Learning and Deep Learning for IoT malware detection and classification. We will discuss the different methods for classifying and detecting IoT malware. In [5], the authors have used fuzzy and fast fuzzy pattern trees for malware detection and classification. The proposed method transforms the Opcodes into vector spaces and uses fuzzy classification. The fuzzy pattern tree nodes take descendants' values as input and combine them with fuzzy operators. The output is submitted to its predecessor, and the sample is classified as malware or benign based on the output.

Table 3.1: Comparison of Existing Techniques for IoT Malware Detection

| Ref. | Year | Graph Based or Non-Graph Based | Features Used | Algorithm Used | Task (Detection or Classification) |
|---|---|---|---|---|---|
| [58] | 2019 | Graph-Based | PSI graph | Deep Learning | Detection |
| [6] | 2020 | Graph-Based | PSI rooted subgraph | Machine Learning | Detection |
| [30] | 2018 | Non-Graph-Based | String, statistical and structural features | Machine Learning | Detection and Classification |
| [51] | 2020 | Non-Graph-Based | Printable strings | Machine Learning | Detection and Classification |
| [76] | 2020 | Non-Graph-Based | Byte sequences from the entry point of ELF binary | Machine Learning | Detection and Classification |
| [28] | 2019 | Graph-Based | Control Flow Graph | Deep Learning | Detection |
| [3] | 2018 | Non-Graph-Based | Opcodes | Deep Learning | Detection |
| [39] | 2021 | Non-Graph-Based | String and Image-based features | Deep Learning | Detection and classification |

| | | | | | |
|---|---|---|---|---|---|
| [47] | 2020 | Graph-Based | Control Flow Graph | Deep Graph Convolutional Neural Network | Detection |
| [24] | 2022 | Graph-Based | Control Flow | Graph Deep Learning | Detection and classification |

In [52], the proposed approach takes byte n-grams from PE binaries and uses the most frequent n-grams as features. Furthermore, the proposed method uses association mining, where mining the n-gram features gives association rules that satisfy minimum support and minimum confidence constraints. The classifier can detect if a PE file is malicious or benign based on the association mining rule. The proposed method is compared with machine learning classifiers such as Naive Bayes, KNN and SVM. However, the proposed method outperforms the other classifiers in terms of accuracy.

The authors have proposed different frameworks or combined techniques for IoT malware detection and classification in other research work. For example, In [70], the authors propose combining machine learning and deep learning techniques for zero-day malware, and in [31], the authors introduce a binary static analysis framework. This framework is used for detection and performs context-sensitive and flow-sensitive analysis to analyze various architecture binaries.

Most of the recent research works have focused on using centralized learning techniques. However, the centralized learning techniques share confidential user data with a central server. Here, in this thesis, we propose to use a decentralized learning technique called Federated Learning. In the following sections, we will discuss the advantages of integrating Federated Learning and IoT and the detailed workflow of Federated Learning.

## 3.3 Advantages of Integrating Federated Learning and IoT

There are several benefits to integrating FL for IoT malware analysis; in this section, we will discuss them in detail.

1. *Data Privacy and Security*

   To understand the pattern of data, train the data and get insights, centralized learning techniques such as ML and DL algorithms are used. In these techniques, the data of different businesses in different locations are sent to a central server where all the data are stored and trained. As the IoT application's user data can be sensitive and contain sensitive user information, centralized techniques can potentially expose data to potential attackers and intruders. In Federated learning, the sensitive user data is not transferred to any central location for training the algorithm but stays on the IoT device, and only the parameters of the model are shared with a central server for collaborative learning.

2. *Improved Network Performance*

   IoT devices require a huge network infrastructure to communicate and handle the data generated from these devices. This potentially affects the performance of the network. In FL, since user data is present in the IoT device and not transferred to a central server, the traffic in the network is reduced. This increases the overall performance of the network.

3. *Scalability*

   The conventional ML algorithms fail to scale to the massive amount of data generated from IoT devices. The integration of FL with IoT enables it to scale the learning as it is not required to train large volumes of data but focuses on the aggregation of model parameters. This improves the scalability of the FL over the other centralized techniques available.

## 3.4 Workflow of Federated Learning

In this section, we discuss the workflow of Federated Learning and the topology of Federated Learning can be seen in Figure 3.8.

- *Local Model Training*: The local model training occurs in each individual device where the model is fetched with the local data. After the model is trained with the respective local data, a local model is generated.

- *Local Model Updates*: After the generation of the local model, each model update contains the weights and parameters from each device. These updates are then sent to the aggregator.

- *Aggregating the Global model*: After receiving these updates from every local model, the global model aggregates each of the updates received by executing aggregation algorithms such as Federated Averaging (FedAvg).

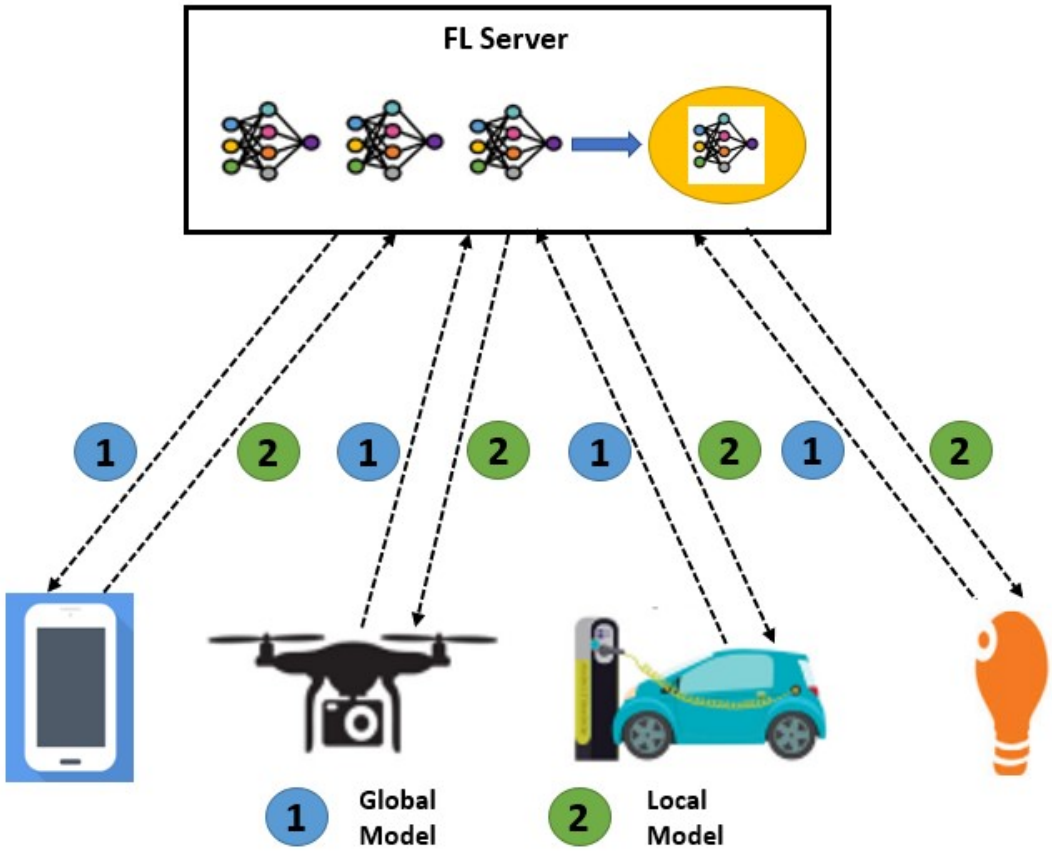


Figure 3.8: Topology of Federated Learning

- *Global Model Generation* After the global model is generated by averaging the local model updates, the global models are sent back to the local models. In this way, the individual devices learn collaboratively from a shared model. There are various categories of FL based on the ways of splitting the data. The categories of FL include Horizontal FL, Vertical FL, and Hybrid FL.

  - *Horizontal FL* - The horizontal FL contains different data samples but shares the same feature space. Some examples of Horizontal FL are Next-word prediction, recommendation systems, and wake-word detectors [14]. In [12], the authors have proposed an algorithm to achieve fairness and accuracy to reduce the uneven distribution of data across horizontal FL.

  - *Vertical FL* - The Vertical FL shares different sample spaces, but the sample IDs are the same. For example, a grocery store and a bank in the same area might have similar customers, but their business structure (feature space) is different. Since a large amount of data generated from these systems are often low quality, in [11], the authors propose an explainable vertical FL to reduce the computational complexity.

  - *Federated Transfer Learning (FTL)* - The FTL combines vertical and horizontal FL. FTL is applicable when the data samples and feature spaces differ in two clients' datasets [14]. FTL is applied to handle variance in data samples and feature space while performing on-device learning. In [21], the authors propose a semi-supervised FTL to reduce model overfitting due to insufficient overlapping training samples in FL scenarios. Here the proposed method uses non-overlapping samples from all parties to expand the training set for each party to improve local model perfor-

mance. In [13], the authors propose using FTL for industrial missing data imputations where the knowledge is indirectly transferred to the target edge through helper models.

## 3.5 Concluding Remarks

This section summarizes all the recent research work for IoT malware detection and classification. We discussed the works related to graph- and non-graph-based features in static analysis. Major research works have concentrated on extracting either graph-based or non-graph-based features. The graph-based features can generalize and represent structured and complex information about the behaviour of malware. In comparison, the non-graph-based features contain properties of a binary file structure and take lesser time for feature extraction to classify an executable file as malicious or benign. In this thesis, we combine the strengths of graph-based and non-graph-based features for the feature extraction process.

Also, as we have seen, most of the research works have focused on using centralized learning techniques leveraging Machine Learning and Deep Learning techniques. The centralized learning technique sends the data from the IoT devices to a central server for training the algorithm. However, this causes several security concerns as they share confidential user data for training in the model. Therefore, in this thesis, we propose to use a decentralized learning technique known as Federated Learning. In this, the sensitive user data is not moved off the device. Instead of transferring the data, the local models present at each device and the global model share the model parameters at the end of each Federated Learning process.

At the end of this chapter, we have discussed the advantages of integrating Federated Learning and IoT and presented the detailed workflow of Federated Learning. The following chapter 4 presents the proposed model in detail.

# Chapter 4

# Proposed Method

## 4.1 Overview

This chapter proposes a static analysis for IoT malware detection using Federated Learning. The proposed framework can be used for efficiently detecting IoT malware samples while preserving the security of confidential user data by not moving it off the device for training purposes, as seen in centralized learning techniques. This can help several industries protect their sensitive user information.

Our initial discussion focuses on the motivation for the proposed method as well as the drawbacks of previous research works. Here, we highlight the crucial importance of our proposed method. Next, in the proposed method, we provide an overview of the various components of our proposed work.

Later, we provide a detailed architecture of the proposed two-stage model for IoT malware detection using Federated Learning. The two-stage architecture consists of feature extraction, obtaining the ideal parameters for the global model, and using Federated Learning to detect IoT malware and benign samples.

## 4.2 Motivation

The previous research works have shown the rapidly growing importance of malware detection as the number of IoT devices used across different industries is on the rise. With the increasing number of IoT devices in industries, they are becoming an easy target for attackers to exploit their vulnerabilities. Most previous research works have focused on using Machine Learning and Deep Learning. However, most of them are centralized learning techniques. In the centralized learning techniques, the data from the IoT devices are shared with a central server for training the model. As a result, sharing the user data with a centralized server and receiving back the updated model from the centralized servers causes several security concerns, as they do not efficiently protect confidential user data.

Therefore, it is crucial to protect the security of confidential user data by using emerging decentralized techniques such as Federated Learning. This research proposes a Federated Learning-based approach that employs a Random Forest model for detecting IoT malware samples. Here, the proposed architecture is a two-stage approach. In the first stage, we extract graph-based and non-graph-based features from the ELF binary files, perform feature selection techniques, and build a Random Forest model with ideal parameters. This Random Forest model acts as the global model for the Federated Learning process in the second stage of the proposed architecture.

Our proposed framework addresses the problem of protecting the security of the user data without moving off the local data from IoT devices for training the model. Unlike centralized learning techniques, where the data is shared with a central server, in Federated Learning, only the model parameters are transferred between the local and global models.

## 4.3 Proposed Method

The proposed approach has two stages. The first stage consists of feature extraction and selection techniques, the FL global model, and the hyperparameter tuning technique. The second module consists of the Federated Learning technique along with their clients. In this section, we will discuss the components of the proposed approach in detail.

- *Data Collection Preprocessing:* The ELF binaries are obtained from the user, preprocessed, and given as input to the feature extraction module.

- *Feature Extraction:* The proposed 30 features are obtained, including 15 graph-based features and 15 non-graph-based features from each ELF binary file. The extracted feature passes through a feature selection technique to receive the top 20 best features.

- *Machine Learning Model:* The extracted and selected features are passed through a machine learning model. This model is used as the global model for the federated learning rounds.

- *Federated Learning clients:* The initial global model is the generated machine learning model in the previous component, along with the obtained parameters. The FL clients are the IoT devices that contain their confidential user data and share only their model parameters with the global model.

- *Prediction Results:* The prediction result is a binary output containing whether there is malicious or benign activity.

Next, we will discuss each of the stages in the architecture in detail.

### 4.3.1 Proposed Architecture - Stage I

The first stage of architecture consists of data collection, preprocessing, feature extraction, feature selection and building the model with hyperparameter tuning to obtain the ideal parameters. Figure 4.1 represents the stage I architecture. In detail, we will discuss the components present in stage-I of architecture as follows.

1. *Data Collection:* First, in the data collection, we have the ELF binary files containing both malicious and benign samples. These ELF binary files are obtained from the user and given as input to the preprocessing module.

2. *Preprocessing:* In preprocessing phase, we first disassemble the ELF binary files using radare2 [73]. After disassembling the ELF binary files, we proceed to the feature extraction phase.

3. *Feature Extraction:* After disassembling the ELF binary files using radare2 [73], we extract two types of features, graph-based and non-graph-based features, using static analysis. We extract 30 features, including 15 graph-based and 15 non-graph-based features.

4. *Feature Selection:* After extracting the 15 graph-based and 15 non-graph-based features, we perform feature selection using Random Forest feature importance to obtain the top 20 features to feed into the Machine Learning model.

5. *Machine Learning Model:* The Machine Learning model used here is a Random Forest Model. We perform hyperparameter tuning to obtain its ideal parameters. This generated Random Forest model acts as the global model in the next stage of the architecture.

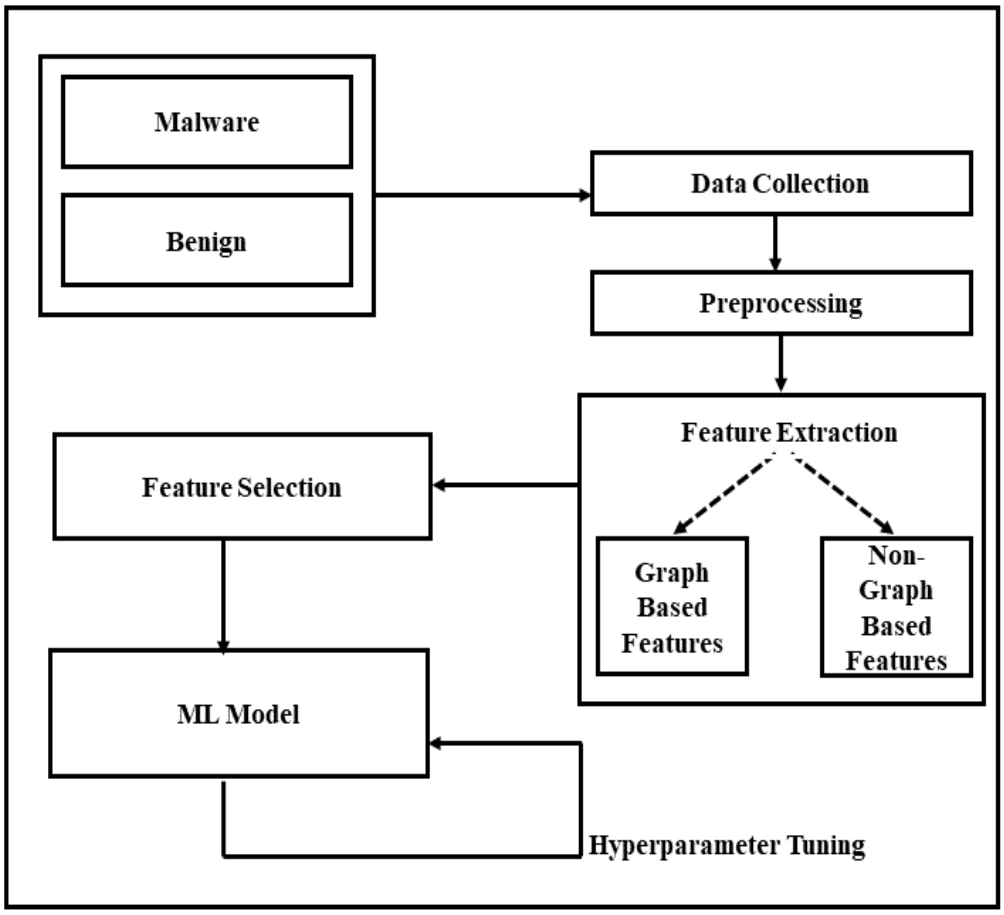


Figure 4.1: Proposed Architecture Stage-I

### 4.3.2 Proposed Architecture - Stage II

The second stage of the proposed model is where we implement the Federated Learning process. In chapter 3, we have shown the overview and topology of Federated Learning (FL). First, we will discuss in detail the various components present in FL, as presented in Figure 4.2 for our proposed approach in this section, followed by the workflow for the second stage.

1. *Local Model Training* - First, each IoT device in the network contains its individual models. These models are known as local models. Then, each of these local models is fetched with its own local confidential user data obtained from the IoT device. After the training is completed, a local model is generated.

2. *Local Model Generation* - After local model training, the generated local model from all the participating IoT devices in the network, the local model parameters are shared with the global model.

3. *Aggregating Model parameters* - In the global model, after receiving all the model parameters from the local model from the participating IoT devices in the network, it uses aggregation techniques to generate a global model. This global consists of all the aggregated parameters from the local model.

4. *Updated Global Model Parameters* - After generating a global model by aggregating the local model parameters, the updated global model parameters are shared with the local model for the next round of the FL process.

Next, we will discuss the workflow for the second stage of our proposed approach. We first receive the generated Machine Learning model from the first stage. Then, each IoT device in the network receives these model parameters for its initial round of the Federated Learning (FL) process. Hence, for the first round of the FL process, the received model parameters from the first stage are the global model, distributed to the local models for their first round of the FL process.

Then, for the first round of the FL process, each IoT device in the network feeds its local IoT device data to the local model with the received initial parameters. Next, each local model is trained with the individual IoT device data and generates a local model. Finally, the local model parameters are shared with the global model, which performs aggregation techniques to aggregate the received local model parameters from all the IoT devices in the network.

After aggregating the local model updates from all the IoT devices in the network, it generates a new global model based on aggregating the received parameters. Finally, the updated global model parameters are shared with all the IoT devices in the network for the next round of the FL process.

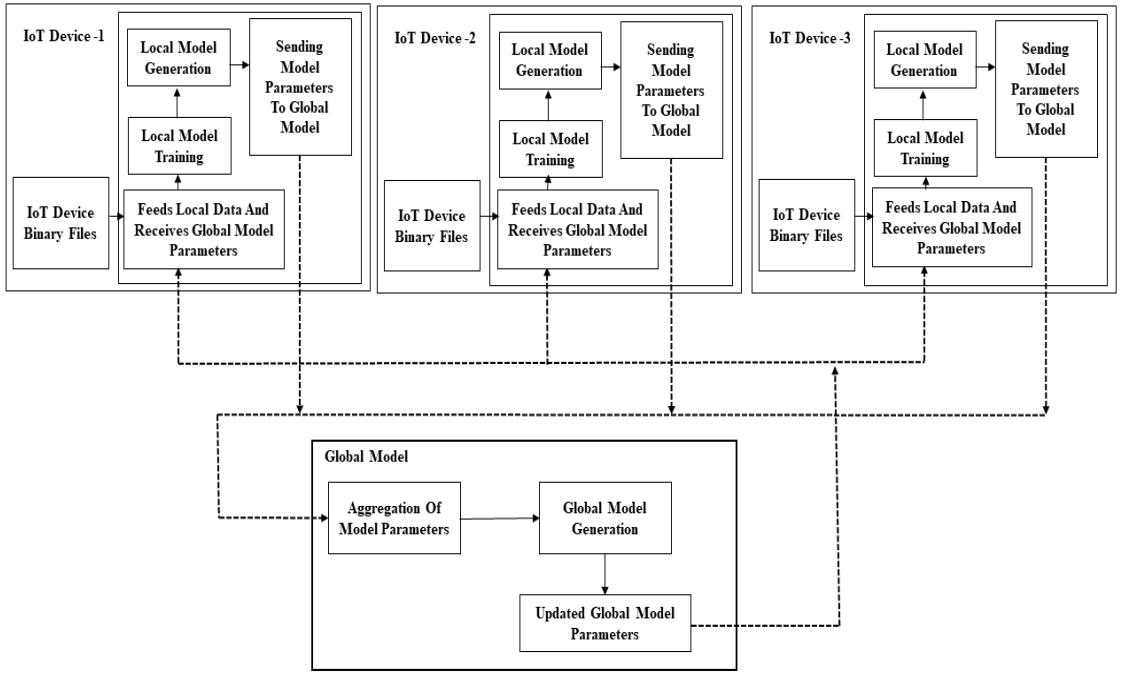


Figure 4.2: Proposed Architecture Stage-II

## 4.4 Concluding Remarks

In this chapter, we discussed several thesis contributions listed in Chapter 1. First, we discussed the motivation behind the proposed method, discussing in detail the drawbacks of the previous research works. Then, we discuss the two-stage proposed architecture and the integration of Federated Learning and IoT for IoT malware detection. In stage I, we discussed in detail the extraction of graph-based and non-graph-based features, the feature selection and building the Machine Learning model with ideal parameters.

Finally, in stage II, we discuss the local and global models for the initial Federated Learning process obtained from the previous stage. The Federated Learning process and its parameter aggregation and model update techniques are also discussed in detail. Finally, in chapter 5, we will discuss the experiments and results of the proposed framework.

# Chapter 5

# Experiments & Results

## 5.1 Overview

This chapter starts by giving an overview of the experimental setup and the dataset used for the experiments and obtaining the results. Later, we discuss the features used for the proposed method in detail by listing all the features and providing a detailed discussion of the extraction process. As discussed in chapter 3, where most of the previous research works focus on using centralized learning techniques, here we show the experimentation and results of the proposed decentralized Federated Learning technique.

Later in the chapter, we present in detail the implementation results of the proposed models and provide a comparison of the proposed Federated Learning model with a non-Federated baseline model. Finally, the results from the experiments prove the effectiveness of the proposed system.

## 5.2 Experimental Setup

The implementation and execution of all the modules were processed on Windows 10, 8 GB RAM with NVIDIA Geforce GTX 1650. All the data preprocessing and feature extraction were implemented using Python scripts and Python public libraries. In addition, the FL was implemented using the Flower open-source library [35].

## 5.3 Dataset

We used ELF binary files from the IoTPoT dataset [61] and online repositories for the experiments. The IoTPoT dataset consists of 4,000 malware executable files captured for one year between 2016/10/02 and 2017/10/02 and 1276 benign samples collected from online repositories. After the features were extracted from these ELF binary files, we created a CSV file with the extracted features and used them as the input to the proposed model.

## 5.4 Features

Here, we discuss the proposed method based on the extracted graph-based and non-graph-based features. We have used both graph-based and non-graph-based features, as the non-graph-based features contain the properties of an ELF binary file. Whereas the graph-based features can generalize and represent structural and complex information about the behaviour of the malware sample. We have used radare2 [73] to disassemble the ELF binary files.

For the feature extraction process in the proposed approach, we extract 30 features, including 15 graph-based and 15 non-graph-based features. For the non-graph-based features, we have used a combination of string, structural and statistical features. For the graph-based features, we extract a Function Call Graph (FCG) from the ELF binary files to analyze the graph-based features. The FCG is a control flow graph representing a computer program's calling relationship between the subroutines. Each node represents a function, and an edge here indicates the function calls. First, we obtain the Function Call Graph (FCG), as shown in Figure 5.1.

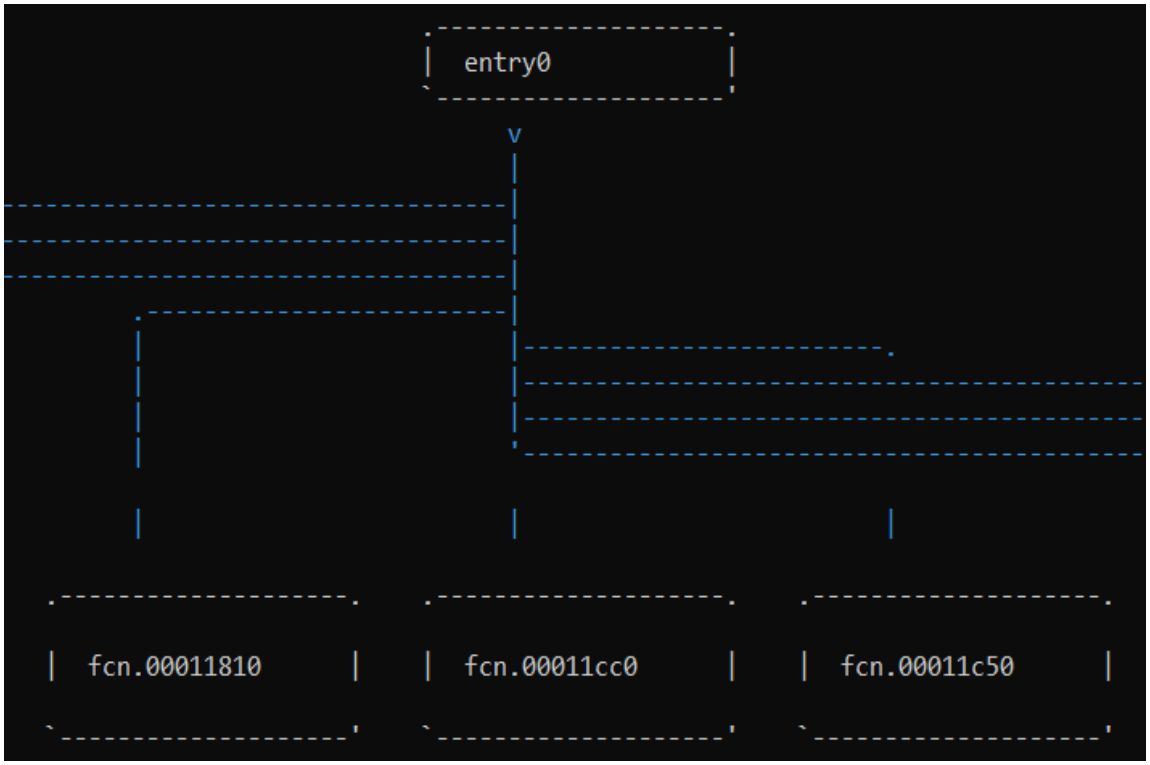


Figure 5.1: A Part of Function Call Graph in ELF Binaries

After getting the FCG, we converted the obtained graph to a Graphviz dot file for the ELF binary files, as shown in Figure 5.2. For obtaining the graph and Graphviz dot file, radare2 [73] was used. After obtaining the Graphviz dot file, we process it using python scripts to extract the features. The list of extracted features is presented in Table 5.1.

```
"0x00095920" [label="fcn.00095920" URL="fcn.00095920/0x00095920"];
"0x0003e930" -> "0x000b986c" [color="#61afef" URL="fcn.000b986c/0x000b986c"];
"0x000b986c" [label="fcn.000b986c" URL="fcn.000b986c/0x000b986c"];
"0x0003e930" -> "0x000c7d28" [color="#61afef" URL="fcn.000c7d28/0x000c7d28"];
"0x000c7d28" [label="fcn.000c7d28" URL="fcn.000c7d28/0x000c7d28"];
"0x0003e930" -> "0x000439c0" [color="#61afef" URL="fcn.000439c0/0x000439c0"];
"0x000439c0" [label="fcn.000439c0" URL="fcn.000439c0/0x000439c0"];
"0x0003e930" -> "0x0001f2e0" [color="#61afef" URL="fcn.0001f2e0/0x0001f2e0"];
"0x0001f2e0" [label="fcn.0001f2e0" URL="fcn.0001f2e0/0x0001f2e0"];
"0x0003e930" -> "0x00147bb0" [color="#61afef" URL="fcn.00147bb0/0x00147bb0"];
"0x00147bb0" [label="fcn.00147bb0" URL="fcn.00147bb0/0x00147bb0"];
"0x001487e0" [label="aav.0x001487e0" URL="aav.0x001487e0/0x001487e0"];
"0x001487e0" -> "0x00147bd0" [color="#61afef" URL="fcn.00147bd0/0x00147bd0"];
"0x00147bd0" [label="fcn.00147bd0" URL="fcn.00147bd0/0x00147bd0"];
"0x001487e0" -> "0x00148714" [color="#61afef" URL="fcn.00148714/0x00148714"];
"0x00148714" [label="fcn.00148714" URL="fcn.00148714/0x00148714"];
"0x001487e0" -> "0x00149690" [color="#61afef" URL="fcn.00149690/0x00149690"];
"0x00149690" [label="fcn.00149690" URL="fcn.00149690/0x00149690"];
"0x001487e0" -> "0x00147c90" [color="#61afef" URL="fcn.00147c90/0x00147c90"];
"0x00147c90" [label="fcn.00147c90" URL="fcn.00147c90/0x00147c90"];
"0x001487e0" -> "0x00147bb0" [color="#61afef" URL="fcn.00147bb0/0x00147bb0"];
"0x00147bb0" [label="fcn.00147bb0" URL="fcn.00147bb0/0x00147bb0"];
"0x001487e0" -> "0x00147e14" [color="#61afef" URL="fcn.00147e14/0x00147e14"];
"0x00147e14" [label="fcn.00147e14" URL="fcn.00147e14/0x00147e14"];
"0x001487e0" -> "0x0014f610" [color="#61afef" URL="fcn.0014f610/0x0014f610"];
"0x0014f610" [label="fcn.0014f610" URL="fcn.0014f610/0x0014f610"];
"0x001487e0" -> "0x0014b558" [color="#61afef" URL="fcn.0014b558/0x0014b558"];
"0x0014b558" [label="fcn.0014b558" URL="fcn.0014b558/0x0014b558"];
"0x001487e0" -> "0x0014a344" [color="#61afef" URL="fcn.0014a344/0x0014a344"];
```

Figure 5.2: Function Call Graph Graphviz dot in ELF Binaries

Table 5.1: List of Features

| Graph-Based Features | Non-Graph-Based Features |
|---|---|
| Number of Nodes | Total Number of Functions |
| Number of Edges | Total Number of Instructions |
| Density | Number of Binary Arithmetic Instructions |
| Number of Connected Components | Number of Decimal Arithmetic Instructions |
| Is Connected | Number of Logical Instructions |
| Maximum degree | Number of Data Transfer Instructions |
| Minimum degree | Number of Control Transfer Instructions |
| Is Empty | Number of Segments |
| Is Weighted | Number of Call Instructions |
| Closeness Centrality | Size of Binary Program |
| Mean Degree | Length of Byte Instructions |
| Median Degree | Total Number of Imports |
| File Size | Total Number of Exports |
| Load Centrality | Total Number of Flags |
| Degree Centrality | Total Number of String Instructions |

We discuss each of the extracted features below in detail. First, we discuss the graph-based features in detail, and then we move on to the non-graph-based features:

1. *Number of Nodes*- The number of nodes is the general characteristic used to highlight the structural size of the binary in terms of the nodes present in the graph. This represents the total number of nodes in a graph.

2. *Number of Edges* - The number of edges is the general characteristic used to highlight the structural size of the binary in terms of the edges present in the graph, representing the total number of edges present in a graph.

3. *Density* - The density of a graph, *G = (V, E)*, is the closeness of all its edges to the maximum number of edges. The density of *G* can be presented as the average normalized degree. Below is the equation to calculate the density of a graph:
$$Density = \frac{1}{n}\sum_{i=1}^{n}\frac{deg(v_i)}{n \quad 1}, V = v_1, v_2, \ldots, v_n$$

4. *Number of Connected Components* - In graph *G*, a connected component is a sub-graph in which the vertices are connected but not connected to other vertices in the sub-graph. The number of components of *G* is the cardinality of a set that contains such components.

5. *Is Connected* - The Is Connected feature returns a boolean value to indicate whether the graph, *G = (V, E)*, is connected.

6. *Maximum Degree* - For a graph, *G = (V, E)*, the maximum degree is the degree of the vertex with the greatest number of edges incident to it.

7. *Minimum Degree* - For a graph, $G = (V, E)$, the minimum degree is the degree of the vertex with the least number of edges incident to it.

8. *Is Empty* - For a given graph, $G = (V, E)$, the Is Empty feature indicates with a boolean value if the graph is empty or not.

9. *Closeness Centrality* - The closeness centrality of a graph indicates how close a node is to all the other nodes in the network. It can be calculated as the average of the shortest path length from the node to every other node in the network.

10. *Is Weighted* - The Is Weighted feature returns a boolean value signifying whether a graph, $G = (V, E)$, is weighted.

11. *Mean Degree* - The mean degree is the average number of edges per node in the graph. It is presented by dividing the total number of edges by the total number of nodes.

12. *Median Degree* - The median degree is obtained after acquiring the degree of the graph, where it is the summation of all the degrees of the nodes divided by the total number of nodes.

13. *File Size* - The file size indicates the size of the dot file obtained after extracting from the Function Call Graph using radare2 [73]

14. *Load Centrality* - The load centrality of a node is the fraction of all shortest paths that pass through that node.

15. *Degree Centrality* - The degree centrality for a node v is a count of the number of edges that connect to it.

16. *Total Number of String Instructions* - This represents the total number of all types of strings present in an ELF binary file.

17. *Total Number of Functions* - This represents all the functions present in an ELF binary file.

18. *Number of Control Transfer Instructions* - The control transfer instructions transfer the flow of execution of the program to a new address specified in the instruction directly or indirectly. This feature represents all the control transfer instructions in an ELF binary file, such as call, enter, loop etc.

19. *Number of Binary Arithmetic Instructions* - The binary arithmetic instructions perform basic binary integer computations on byte words located in memory or general-purpose registers. These include add, adc, div, mul, etc.

20. *Number of Decimal Arithmetic Instructions* - The decimal arithmetic instructions perform decimal arithmetic operations such as aaa, aad, aam, etc.

21. *Number of Segments* - This represents the segment register instructions that load segment addresses into the registers, and some examples of these instructions include lds, les etc.

22. *Number of Data Transfer Instructions* - The data transfer instructions consist of operations such as conditional moves, stack access and data conversion.

23. *Size of Binary Program* - This represents the size of the ELF binary file.

24. *Total Number of Flags* - This represents the total number of flags in the ELF binary file.

25. *Number of Logical Instructions* - These instructions are the ones that perform basic logical operations such as AND, OR, etc.

26. *Total Number of Instructions* - The total number of instructions includes the summation of all types of instructions present in the ELF binary file.

27. *Total Number of Imports* - This represents the total number of imports in the ELF binary file.

28. *Total Number of Exports* - This represents the total number of exports in the ELF binary file.

29. *Length of Byte Instructions* - This analyzes the length of byte instructions for references, and this is done using radare2 [73]

30. *Number of Call Instructions* - This represents all the call instructions in the ELF binary files, such as edx, ecx, etc.

## 5.5 Finalizing the Proposed Model

In this section, we will discuss the proposed model in detail. After extracting and selecting the features using feature importance, we build the Random Forest model and perform hyperparameter tuning using the Grid Search technique to obtain the top 20 parameters. Here, for experimentation purposes, we have used 3 Federated Learning clients, the IoT devices with their local binary files for each device. In the initial FL round for the global model and three local models, we use the model parameters obtained from building the Random Forest model in the previous stage after feature extraction and selection.

After the first FL round, each device's local model parameters are aggregated at the global model using FedAvg. Then, the aggregated global model parameters are sent to all the local models for the next round. A similar process is repeated until the number of FL rounds is completed. In this experiment, we have processed the model for 3 FL rounds. At the end of each round, we receive an accuracy for the FL process.

### 5.5.1 Performance Metrics

The following are the criteria used to evaluate the proposed method's performance measures.

1. *True Negative (TN):* Number of benign data detected as a non-malicious activity.
2. *False Negative (FN):* Number of malicious activity not detected as a non-malicious activity.
3. *True Positive (TP):* Number of malicious attacks detected as malicious activity.
4. *False Positive (FP):* Number of benign data detected as malicious activity.

**Accuracy** is defined as the total number of correct predictions and is calculated as follows:

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

**Precision** is the ratio of true detection of malicious attacks over the sum of true malicious attacks and false detection of benign data and is calculated as follows:

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

**Recall** is the ratio of true positives over the sum of true malicious detection and false detection of benign data and is calculated as follows:

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

**F1-measure** is the weighted average of the precision and recall, defined as follows:

$$F_1 = \frac{2 \quad \text{TP}}{2 \quad \text{TP} + \text{FP} + \text{FN}}$$

## 5.6 Experimental Results

In our first round, we received an accuracy of 87%. Later in the second round, we received an accuracy of 90%, and in the final round, we received an accuracy of 95%. In Table 5.2, we have presented the accuracy at the end of each round of the FL process. Figure 5.3 presents the confusion matrix representing the True Positive, False Positive, True Negative, and False Negative. The results show that the Federated Learning process can achieve high accuracies for classifying the malware and benign samples while preserving the security of confidential user data.

Table 5.2: Federated Learning Rounds with Accuracy

| Federated Learning Round | Accuracy |
|---|---|
| Round 1 | 87% |
| Round 2 | 90% |
| Round 3 | 95% |

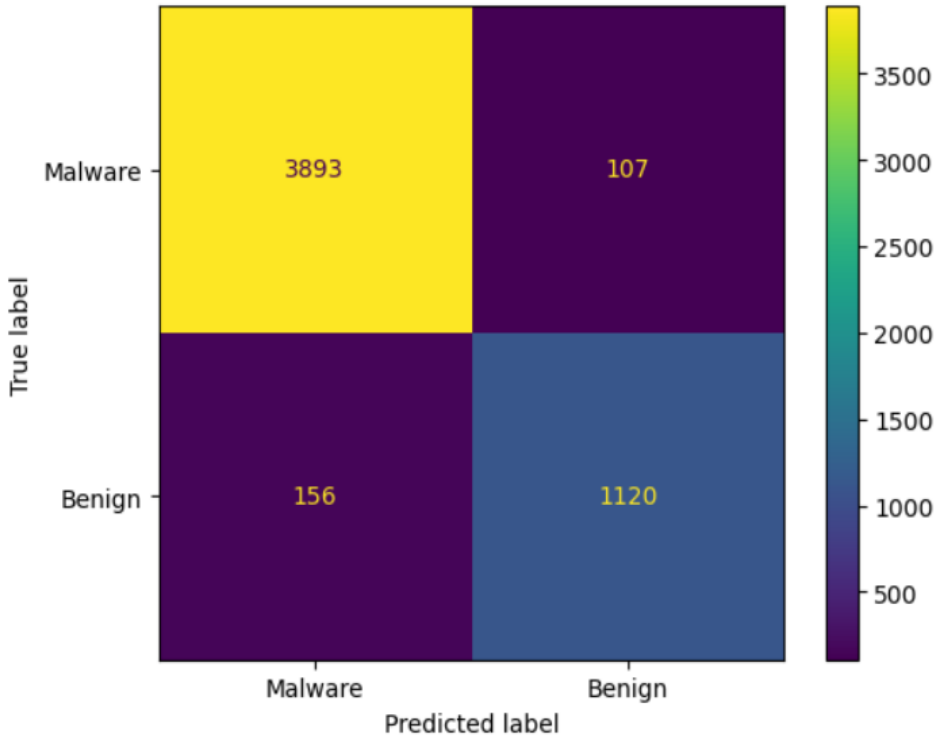


Figure 5.3: Confusion Matrix Results

## 5.7 Comparing Proposed Model with Non-Federated Baseline model

The comparison mainly focuses on existing work on IoT malware detection using a centralized technique such as Deep Learning and the proposed Federated Learning model. In the paper [57], the authors proposed using a CNN for IoT malware detection. However, the baseline method has used a centralized learning technique. In comparison, the proposed method uses the decentralized Federated Learning technique and protects the security of confidential user data. Furthermore, in terms of accuracy in classifying the malware and benign samples, the proposed method showed improved overall accuracy and performance in detection compared to the centralized learning technique.

Figure 5.4 shows the accuracy of classifying the IoT samples as malicious or benign using the proposed approach versus the non-FL baseline model. As can be seen from the figure, the accuracy of the federated approach has an accuracy of 95%, whereas the accuracy of the non-FL baseline reached 92%. Lastly, Table 5.3 compares the proposed method's accuracy, precision, recall, and F measure and the non-FL baseline model.

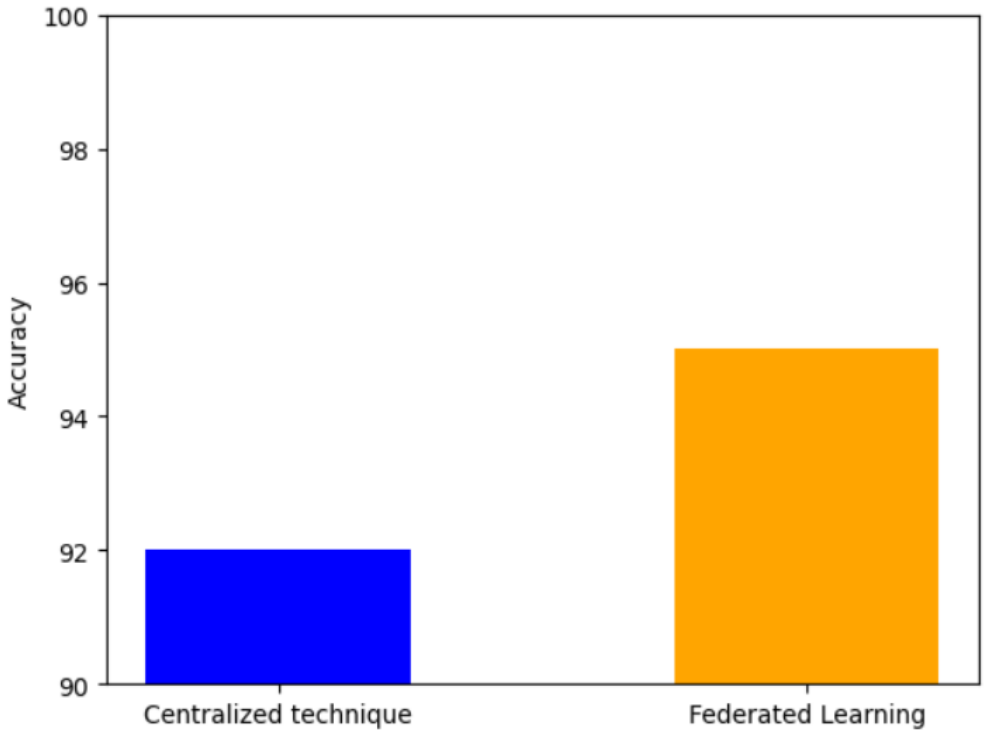


Figure 5.4: Accuracy of Proposed Federated Learning method Vs. Non-Federated Baseline method

Table 5.3: Comparison Between Proposed Method and Baseline Method

| Method | Acc. | Pre. | Rec. | F1-measure |
|---|---|---|---|---|
| Proposed FL Method | 95% | 96% | 97% | 96.4% |
| Non-FL Baseline Method | 92% | 91.7% | 96.3% | 94% |

## 5.8 Concluding Remarks

This chapter evaluated our proposed method in terms of effectiveness and efficiency based on the IoTPot dataset. First, we have presented the experimental setup and a detailed description of the dataset used for the experiments. Then, we have also discussed the features and the feature extraction process in detail, as introduced in chapter 4. Furthermore, after that, we discussed the Federated Learning process in detail with further information explaining the global and local models along with the number of rounds of the Federated Learning process and presented the accuracies at the end of each round.

Finally, we have also discussed and compared the result of the proposed decentralized Federated Learning model with a centralized non-Federated baseline model. The results show the efficiency and accuracy of the proposed model over the previous research work while protecting confidential user data and detecting IoT malware efficiently in the network.

# Chapter 6

# Conclusion & Future Works

## 6.1 Conclusions

As several industries begin to widen the usage and diversification of IoT devices, there is an increase in the risk of new attacks on these networks. Hence, it is essential to protect the security of the user data and preserve their confidential information effectively and efficiently. In this thesis, we first review previous research on IoT malware detection, providing a detailed survey of the different techniques and approaches used for IoT malware detection. Most of the previous research works have focussed on using centralized techniques where confidential user data is transferred from the IoT device to a central server for training algorithms, which compromises the security of the user data.

As a solution to this problem, in this thesis, we proposed Federated Learning, a decentralized learning technique in which user data is not moved off from the IoT devices. The proposed method consists of a two-stage architecture, where we use static analysis for feature extraction in the first stage. We have extracted 15 graph-based and 15 non-graph-based features using radare2 [73] from the ELF binary files

obtained as input. Then, we perform feature selection techniques, build a Machine Learning model, and obtain their ideal parameters by performing hyperparameter tuning techniques. In the second stage of the architecture, for the Federated Learning approach, we used the obtained Machine Learning model from the previous stage as the initial model for the initial federated rounds. Moreover, this method is compared with previous centralized learning techniques. The comparison results prove that the proposed method performs better in detecting malware and benign samples and protects the security of confidential user data.

## 6.2 Future Works

1. Resource-Constrained IoT devices: The next step would be to consider techniques for resource-constrained IoT devices, as some of them can have low bandwidth, power capabilities and limited storage. Utilizing resource-aware training, virtual workers and edge devices could help resolve this issue and expand the Federated Learning ability.

2. Enhanced node selection techniques: As all the IoT devices in the network may not be online at all times and might stay offline for frequent periods due to connectivity or power issues, node selection techniques must be efficient so that the global model aggregation process is not delayed.

3. Server-side attacks: In server-side attacks, the model updates can be tampered by the attackers, and the attacker may try to steal the model updates from the cloud resulting in inconsistency and noise. Using privacy-preserving techniques such as homomorphic encryption and differential privacy techniques may help resolve this concern.

4. Client-side attacks: In client-side attacks, the client-side servers are prone to model-poisoning attacks and data-poisoning attacks. In a model poisoning attack, the attacker might upload poisoned updates, leading to performance degradation and classification errors. In data poisoning attacks, the attackers infiltrate and enter misleading information, tampering with the training of the models. Utilizing blockchain technology to verify each of the model updates might prevent client-side attacks.

# Bibliography


[1] *Chapter 4 - feeling bluetooth: From a security perspective*, Advances in Computers, vol. 81, Elsevier, 2011, pp. 161–236.

[2] *Integrated static and dynamic analysis for malware detection*, Procedia Computer Science **46** (2015), 804–811, Proceedings of the International Conference on Information and Communication Technologies, ICICT 2014, 3-5 December 2014 at Bolgatty Palace  Island Resort, Kochi, India.

[3] *A deep recurrent neural network based approach for internet of things malware threat hunting*, Future Generation Computer Systems **85** (2018), 88–96.

[4] *Early-stage malware prediction using recurrent neural networks*, Computers Security **77** (2018), 578–594.

[5] *Fuzzy pattern tree for edge malware detection and categorization in iot*, Journal of Systems Architecture **97** (2019), 1–7.

[6] *Psi-rooted subgraph: A novel feature for iot botnet detection using classi er algorithms*, ICT Express **6** (2020), no. 2, 128–138.

[7] *A survey of iot malware and detection methods based on static features*, ICT Express **6** (2020), no. 4, 280–286.

[8] *Ag-iot for crop and environment monitoring: Past, present, and future*, Agricultural Systems **203** (2022), 103497.

[9] *Deep learning based cross architecture internet of things malware detection and classi cation*, Computers Security **120** (2022), 102779.

[10] *Deep malware detection framework for iot-based smart agriculture*, Computers and Electrical Engineering **104** (2022), 108410.

[11] *Ev : An explainable vertical federated learning for data-oriented arti cial intelligence systems*, Journal of Systems Architecture **126** (2022), 102474.

[12] *Fairness and accuracy in horizontal federated learning*, Information Sciences **589** (2022), 170–185.

[13] *Fedtmi: Knowledge aided federated transfer learning for industrial missing data imputation*, Journal of Process Control **117** (2022), 206–215.

[14] *Fusion of federated learning and industrial internet of things: A survey*, Computer Networks **212** (2022), 109048.

[15] *Impact of internets of things (iot) in retail sector*, Materials Today: Proceedings **51** (2022), 26–30, CMAE'21.

[16] *Iot data analytics in dynamic environments: From an automated machine learning perspective*, Engineering Applications of Artificial Intelligence **116** (2022), 105366.

[17] *Iot data visualization for business intelligence in corporate nance*, Information Processing Management **59** (2022), no. 1, 102736.

[18] *Iot in healthcare: A scientometric analysis*, Technological Forecasting and Social Change **184** (2022), 122001.

[19] *Light gradient boosting machine with optimized hyperparameters for identi cation of malicious access in iot network*, Digital Communications and Networks (2022).

[20] *Literature review and methodological framework for integration of iot and plm in manufacturing industry*, Computers in Industry **140** (2022), 103688.

[21] *Semi-supervised federated heterogeneous transfer learning*, Knowledge-Based Systems **252** (2022), 109384.

[22] *Static vulnerability mining of iot devices based on control ow graph construction and graph embedding network*, Computer Communications (2022).

[23] *Tools and techniques for collection and analysis of internet-of-things malware: A systematic state-of-art review*, Journal of King Saud University - Computer and Information Sciences (2022).

[24] Ahmed Abusnaina, Mohammed Abuhamad, Hisham Alasmary, Afsah Anwar, Rhongho Jang, Saeed Salem, DaeHun Nyang, and David Mohaisen, *Dl-fhmc: Deep learning-based ne-grained hierarchical learning approach for robust malware classi cation*, IEEE Transactions on Dependable and Secure Computing **19** (2022), no. 5, 3432–3447.

[25] Ahmed Abusnaina, Hisham Alasmary, Mohammed Abuhamad, Saeed Salem, DaeHun Nyang, and Aziz Mohaisen, *Subgraph-based adversarial examples against graph-based iot malware detection systems*, Computational Data and Social Networks (Cham) (Andrea Tagarelli and Hanghang Tong, eds.), Springer International Publishing, 2019, pp. 268–281.

[26] Ahmed Abusnaina, Aminollah Khormali, Hisham Alasmary, Jeman Park, Afsah Anwar, and Aziz Mohaisen, *Adversarial learning attacks on graph-based iot malware detection systems*, 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), 2019, pp. 1296–1305.

[27] Hisham Alasmary, Afsah Anwar, Jeman Park, Jinchun Choi, Daehun Nyang, and Aziz Mohaisen, *Graph-based comparison of iot and android malware*, Com-

putational Data and Social Networks (Cham) (Xuemin Chen, Arunabha Sen, Wei Wayne Li, and My T. Thai, eds.), Springer International Publishing, 2018, pp. 259–272.

[28] 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.

[29] Hisham Alasmary, Aminollah Khormali, Afsah Anwar, Jeman Park, Jinchun Choi, Daehun Nyang, and Aziz Mohaisen, *Analyzing, comparing, and detecting emerging malware: A graph-based approach*, ArXiv **abs/1902.03955** (2019).

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

[31] Hayk Aslanyan, Mariam Arutunian, Grigor Keropyan, Shamil Kurmangaleev, and Vahagn Vardanyan, *Binside : Static analysis framework for defects detection in binary code*, 2020 Ivannikov Memorial Workshop (IVMEM), 2020, pp. 3–8.

[32] Seungyeon Baek, Jueun Jeon, Byeonghui Jeong, and Young-Sik Jeong, *Two-stage hybrid malware detection using deep learning*, Human-centric Computing and Information Sciences **11** (2021), no. 27, 10–22967.

[33] Tao Ban, Ryoichi Isawa, Katsunari Yoshioka, and Daisuke Inoue, *A cross-platform study on iot malware*, 2018 Eleventh International Conference on Mobile Computing and Ubiquitous Network (ICMU), 2018, pp. 1–2.

[34] Emil Bejder, Adam Krog Mathiasen, Michele De Donno, Nicola Dragoni, and Xenofon Fafoutis, *Shake: Shared acceleration key establishment for resource-constrained iot devices*, 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), 2020, pp. 1–6.

[35] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D Lane, *Flower: A friendly federated learning research framework*, arXiv preprint arXiv:2007.14390 (2020).

[36] Abdur Rahim Biswas and Raffaele Giaffreda, *Iot and cloud convergence: Opportunities and challenges*, 2014 IEEE World Forum on Internet of Things (WF-IoT), 2014, pp. 375–376.

[37] Cheng-Yu Chen and Shun-Wen Hsiao, *Iot malware dynamic analysis pro ling system and family behavior analysis*, 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 6013–6015.

[38] 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** (2019).

[39] Mirabelle Dib, Sadegh Torabi, Elias Bou-Harb, and Chadi Assi, *A multi-dimensional deep learning framework for iot malware classi cation and family attribution*, IEEE Transactions on Network and Service Management **18** (2021), no. 2, 1165–1177.

[40] Chris Eagle, *The ida pro book: The uno cial guide to the world's most popular disassembler*, No Starch Press, USA, 2008.

[41] Seoungyul Euh, Hyunjong Lee, Donghoon Kim, and Doosung Hwang, *Comparative analysis of low-dimensional features and tree-based ensembles for malware detection systems*, IEEE Access **8** (2020), 76796–76808.

[42] John Fox, Andrew Donnellan, and Liam Doumen, *The deployment of an iot network infrastructure, as a localised regional service*, 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), 2019, pp. 319–324.

[43] 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), 2018, pp. 2226–2230.

[44] Sibel Gülmez and Ibrahim Sogukpinar, *Graph-based malware detection using opcode sequences*, 2021 9th International Symposium on Digital Forensics and Security (ISDFS), 2021, pp. 1–5.

[45] Raden Budiarto Hadiprakoso, Herman Kabetta, and I Komang Setia Buana, *Hybrid-based malware analysis for e ective and e ciency android malware detection*, 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), 2020, pp. 8–12.

[46] Mehadi Hassen, Marco M. Carvalho, and Philip K. Chan, *Malware classi cation using static analysis based features*, 2017 IEEE Symposium Series on Computational Intelligence (SSCI), 2017, pp. 1–7.

[47] Yakang Hua, Yuanzheng Du, and Dongzhi He, *Classifying packed malware represented as control ow graphs using deep graph convolutional neural network*, 2020 International Conference on Computer Engineering and Application (ICCEA), 2020, pp. 254–258.

[48] Jueun Jeon, Byeonghui Jeong, Seungyeon Baek, and Young-Sik Jeong, *Hybrid malware detection based on bi-lstm and spp-net for smart iot*, IEEE Transactions on Industrial Informatics **18** (2022), no. 7, 4830–4837.

[49] Jueun Jeon, Jong Hyuk Park, and Young-Sik Jeong, *Dynamic analysis for iot malware detection with convolution neural network model*, IEEE Access **8** (2020), 96899–96911.

[50] Ayush Kumar and Teng Joon Lim, *Edima: Early detection of iot malware network activity using machine learning techniques*, 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), 2019, pp. 289–294.

[51] Yen-Ting Lee, Tao Ban, Tzu-Ling Wan, Shin-Ming Cheng, Ryoichi Isawa, Takeshi Takahashi, and Daisuke Inoue, *Cross platform iot-malware family classi cation based on printable strings*, 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2020, pp. 775–784.

[52] Bowei Li, Yongzheng Zhang, Junliang Yao, and Tao Yin, *Mdba: Detecting malware based on bytes n-gram with association mining*, 2019 26th International Conference on Telecommunications (ICT), 2019, pp. 227–232.

[53] Sanjay Madan and Monika Singh, *Classi cation of iot-malware using machine learning*, 2021 International Conference on Technological Advancements and Innovations (ICTAI), 2021, pp. 599–605.

[54] Fahad Mira, *A systematic literature review on malware analysis*, 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), 2021, pp. 1–5.

[55] Quoc-Dung Ngo, Huy-Trung Nguyen, Hoang-Long Pham, Hoang Hanh-Nhan Ngo, Doan-Hieu Nguyen, Cong-Minh Dinh, and Xuan-Hanh Vu, *A graph-based*

*approach for iot botnet detection using reinforcement learning*, Springer-Verlag, 2020.

[56] Quoc-Dung Ngo, Huy-Trung Nguyen, Hoang-Anh Tran, Ngoc-Anh Pham, and Xuan-Hoang Dang, *Toward an approach using graph-theoretic for iot botnet detection*, Association for Computing Machinery, 2021.

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

[58] Huy-Trung Nguyen, Quoc-Dung Ngo, and Van-Hoang Le, *A novel graph-based approach for iot botnet detection*, International Journal of Information Security **19** (2019), 567 – 577.

[59] Khanh Duy Tung Nguyen, Tran Minh Tuan, Son Hai Le, Anh Phan Viet, Mizuhito Ogawa, and Nguyen Le Minh, *Comparison of three deep learning-based approaches for iot malware detection*, 2018 10th International Conference on Knowledge and Systems Engineering (KSE), 2018, pp. 382–388.

[60] Ori Or-Meir, Aviad Cohen, Yuval Elovici, Lior Rokach, and Nir Nissim, *Pay attention: Improving classi cation of pe malware using attention mechanisms based on system call analysis*, 2021 International Joint Conference on Neural Networks (IJCNN), 2021, pp. 1–8.

[61] 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.

[62] Tran Nghi Phu, Le Hoang, Nguyen Ngoc Toan, Nguyen Dai Tho, and Nguyen Ngoc Binh, *C500-cfg: A novel algorithm to extract control ow-based features for iot malware detection*, 2019 19th International Symposium on Communications and Information Technologies (ISCIT), 2019, pp. 568–573.

[63] Tran Nghi Phu, Le Huy Hoang, Nguyen Ngoc Toan, Nguyen Dai Tho, and Nguyen Ngoc Binh, *Cfdvex: A novel feature extraction method for detecting cross-architecture iot malware*, Association for Computing Machinery, 2019.

[64] Prajoy Podder, M. Rubaiyat Mondal, Subrato Bharati, and Pinto Paul, *Review on the security threats of internet of things*, International Journal of Computer Applications **176** (2020), 37–45.

[65] Amit Praseed and P. Santhi Thilagam, *Ddos attacks at the application layer: Challenges and research perspectives for safeguarding web applications*, IEEE Communications Surveys Tutorials **21** (2019), no. 1, 661–685.

[66] Mohammad Abu Qbeitah and Monther Aldwairi, *Dynamic malware analysis of phishing emails*, 2018 9th International Conference on Information and Communication Systems (ICICS), 2018, pp. 18–24.

[67] Aswin Raghuprasad, Suraj Padmanabhan, M Arjun Babu, and P.K Binu, *Security analysis and prevention of attacks on iot devices*, 2020 International Conference on Communication and Signal Processing (ICCSP), 2020, pp. 0876–0880.

[68] Hitesh Kumar Sharma, Anuj Kumar, Sangeeta Pant, and Mangey Ram, *5 application of iot in smart healthcare*, pp. 47–56, 2022.

[69] M. Shobana and S. Poonkuzhali, *A novel approach to detect iot malware by system calls using deep learning techniques*, 2020 International Conference on Innovative Trends in Information Technology (ICITIIT), 2020, pp. 1–5.

[70] Himanshu Kumar Singh, Jyoti Prakash Singh, and Anand Shanker Tewari, *Static malware analysis using machine and deep learning*, Proceedings of International Conference on Computing and Communication Networks (Singapore) (Ali Kashif Bashir, Giancarlo Fortino, Ashish Khanna, and Deepak Gupta, eds.), Springer Nature Singapore, 2022, pp. 437–446.

[71] Jiawei Su, Danilo Vargas Vasconcellos, Sanjiva Prasad, Daniele Sgandurra, Yaokai Feng, and Kouichi Sakurai, *Lightweight classi cation of iot malware based on image recognition*, 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), vol. 02, 2018, pp. 664–669.

[72] Susanto, Deris Stiawan, M. Agus Syamsul Arifin, Mohd. Yazid Idris, and Rahmat Budiarto, *Iot botnet malware classi cation using weka tool and scikit-learn machine learning*, 2020 7th International Conference on Electrical Engineering, Computer Sciences and Informatics (EECSI), 2020, pp. 15–20.

[73] Radare2 Team, *Radare2 github repository*, https://github.com/radare/radare2, 2017.

[74] Chin-Wei Tien, Shang-Wen Chen, Tao Ban, and Sy-Yen Kuo, *Machine learning framework to analyze iot malware using elf and opcode features*, Digital Threats **1** (2020), no. 1.

[75] Igor Vurdelja, Ivan Blažić, Dragan Bojić, and Dražen Drašković, *A framework for automated dynamic malware analysis for linux*, 2020 28th Telecommunications Forum (TELFOR), 2020, pp. 1–4.

[76] Tzu-Ling Wan, Tao Ban, Shin-Ming Cheng, Yen-Ting Lee, Bo Sun, Ryoichi Isawa, Takeshi Takahashi, and Daisuke Inoue, *E cient detection and classi cation of internet-of-things malware based on byte sequences from executable les*, IEEE Open Journal of the Computer Society **1** (2020), 262–275.

[77] Tzu-Ling Wan, Tao Ban, Yen-Ting Lee, Shin-Ming Cheng, Ryoichi Isawa, Takeshi Takahashi, and Daisuke Inoue, *Iot-malware detection based on byte sequences of executable les*, 2020 15th Asia Joint Conference on Information Security (AsiaJCIS), 2020, pp. 143–150.

[78] Shulong Wang, Yibin Hou, Fang Gao, and Xinrong Ji, *A novel iot access architecture for vehicle monitoring system*, 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), 2016, pp. 639–642.

[79] Chia-Yi Wu, Tao Ban, Shin-Ming Cheng, Bo Sun, and Takeshi Takahashi, *Iot malware detection using function-call-graph embedding*, 2021 18th International Conference on Privacy, Security and Trust (PST), 2021, pp. 1–9.

[80] Jiaqi Yan, Guanhua Yan, and Dong Jin, *Classifying malware represented as control ow graphs using deep graph convolutional neural network*, 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2019, pp. 52–63.

[81] Yipin Zhang, Xiaolin Chang, Yuzhou Lin, Jelena Mišić, and Vojislav B. Mišić, *Exploring function call graph vectorization and le statistical features in malicious pe le classi cation*, IEEE Access **8** (2020), 44652–44660.

# Vita

Candidate's full name: Madumitha Venkatasubramanian

University attended :
Bachelor of Computer Science and Engineering
SRM Institute of Science and Technology
2015-2019

Publications:
Madumitha Venkatasubramanian, Saqib Hakak and Arash Habibi Lashkari, **"IoT Malware Analysis using Federated Learning: A Comprehensive Survey"**, resubmitted to IEEE Open Access

Conference Presentations:
Madumitha Venkatasubramanian, Saqib Hakak and Arash Habibi Lashkari, **"Federated Learning Assisted IoT Malware Detection Using Static Analysis"**, ICCNS 2022, the 12th International Conference on Communication and Network Security - Accepted and received the **Best Presentation Award**