DL-FHMC: Deep Learning-based Fine-grained Hierarchical Learning Approach for Robust Malware Classification

Ahmed Abusnaina, Mohammed Abuhamad, Hisham Alasmary, Afsah Anwar, Rhongho Jang, Saeed Salem, DaeHun Nyang, and David Mohaisen,

Abstract—The acceptance of the Internet of Things (IoT) for both household and industrial applications is accompanied by the rapid growth of IoT malware. With the increase of their attack surface, analyzing, understanding, and detecting IoT malicious behavior are crucial. Traditionally, machine and deep learning-based approaches are used for malware detection and behavioral understanding. However, recent research has shown the susceptibility of those approaches to adversarial attacks by introducing noise to the feature space. In this work, we introduce DL-FHMC, a fine-grained hierarchical learning approach for robust IoT malware detection. DL-FHMC utilizes Control Flow Graph (CFG)-based behavioral patterns for adversarial IoT malicious software detection. In particular, we extract a comprehensive list of behavioral patterns from a large dataset of malicious IoT binaries, represented by the shared execution flows, and use them as a modality for malicious behavior detection. Leveraging machine learning and subgraph isomorphism matching algorithms, DL-FHMC provides a state-of-the-art performance in detecting malware samples and adversarial examples (AEs). We start this work by examining the performance of the current CFG-based IoT malware detection systems against adversarial IoT software crafted using Graph Embedding and Augmentation (GEA) techniques. As a result, we show the adversarial capabilities in generating practical functionality-preserving AEs with reduced overhead, highlighting caveats in the state of the current detection systems under adversarial settings. We then introduce suspicious behavior detector, a component that incorporates comprehensive behavioral patterns extracted from three popular IoT malicious families, Gafgyt, Mirai, and Tsunami, and detects AEs with high accuracy, up to 100% under different attack configurations. The suspicious behavior detector operates as a standalone module that can operate alongside other malware detection methods and does not assume prior knowledge of the adversarial attacks nor their configurations.

Index Terms—Adversarial Machine Learning, Deep Learning, Internet of Things, Malware Detection, Adversarial Attacks

1 Introduction

The Internet of Things (IoT) has shown its fast growth in the last decade. The communication media of IoT devices is not limited only in the form of home networking, but also the cellular, which significantly accelerated their connectivity and accessibility. According to Ericsson mobility report [2], 3.5 billion IoT devices are expected to communicate using cellular in 2023. On the one side, taking advantage of a large number of interconnected devices, many applications can be adopted on a large scale. Moreover, due to the high-speed and low-latency connection of these devices, time-critical applications will become feasible in the real world. On the downside, however, high accessibility also provides convenience to adversaries. Due to the constrained resource of IoT devices, the protection function is usually inefficient. Moreover, due to the large population, low physical accessibility, and the unrestricted use of policies in many scenarios, IoT devices can be easily compromised and abused by adversaries to launch a variety of attacks, such as Distributed Denial of Service (DDoS) attacks launched by Mirai botnet [3]. Such adversarial scenarios are expected to grow for many years to come, posing critical and challenging security threats for the IoT ecosystem.

There has been a large body of research work on the problem of malware analysis using both static and dynamic approaches [4], [5], [6], and a few attempts on analyzing IoT malware in particular. Recently, machine learning algorithms, specifically deep learning techniques, are actively utilized for detecting/classifying malicious software from benign ones. However, it should be noted that the research work on IoT malware analysis has been very limited not only in the size of the analyzed samples, but also the utilized approaches [7], [8], [9], [10]. Among the static analysis-based approaches, one of the prominent approaches is to use abstract graph structures for IoT malware analysis and detection, such as the control flow graph (CFG) [11], [12]. Previously, it has been shown that the software graph-based analysis can be incorporated with machine learning methods to introduce more powerful analysis tools [13], [14]. For the IoT malware detection, CFGs allow defenders to extract plentiful feature representations that can be used to distinguish those malware from benign, owing to their various properties, such as the degree distribution, centrality measures, radius, *etc.*. [11]. Those properties can be represented as a feature vector that can be used to enable machine learning algorithms to accurately detect and classify IoT malware samples. Proposed by Alasmary *et al.*, one such application is exploring IoT malware using both graph analysis and machine/deep learning [11]. Their model not only can learn the representative characteristics of the graph, but also can be utilized to build an automatic detection system for predicting the label of the unseen software.

Using machine and deep learning techniques should first address the concerns and challenges related to their security and

- *A. Abusnaina, M. Abuhamad, H. Alasmary, A. Anwar, and D. Mohaisen are with the Department of Computer Science, University of Central Florida, Orlando, FL 32816, USA. H. Alasmary is also with the Department of Computer Science at King Khalid University, Abha 61421, Saudi Arabia. R. Jang is with the Department of Computer Science at Wayne State University, Detroit, MI 48202, USA. S. Salem is with the Department of Computer Science, North Dakota State University, Fargo, ND 58105, USA. D. Nyang is with the Department of Computer Science & Engineering, Ewha Women University, Seoul, South Korea. E-mail: {ahmed.abusnaina,abuhamad,hisham,afsahanwar} @knights.ucf.edu, r.jang@wayne.edu, saeed.salem@ndsu.edu, nyang@ewha.ac.kr, and mohaisen@ucf.edu.*
An earlier version of this work has appeared in IEEE ICDCS 2019 [1].

usability. Recent studies have shown that machine learning-based IoT malware detection methods are prone to adversarial manipulation [15]. Adversarial machine learning has shown the fragile nature of those algorithms to perturbation and data poisoning attacks, leading to misclassification. For example, an adversary can introduce a small modification to the input sample to make the classifier misidentify the malware as benign (*i.e.,* the adversary introduces an AE). Such modification is usually crafted using small perturbation to make the AE undetectable and very difficult to distinguish for the original sample.

Several studies explored the generation of AEs in general image-based classification problems [16], [17] as well as in the context of malware classification [1], [18], [19]. Also, there is an active trend in the research area towards investigating adversarial machine learning to overcome these threats. However, there is very limited research that aims to understand the impact of adversarial learning on deep learning-based IoT malware detection systems and practical implications, especially for those approaches that utilize CFG features. We stress the importance of addressing the threat posed by the machine learning vulnerability to AEs, particularly in security-sensitive applications. We undertake this challenge by ❶ showing the high potential of successful detection/classification of IoT malware using deep learning methods; ❷ assessing the robustness of such methods to AEs generated by different state-of-the-art CFG-based AEs generation techniques; ❸ introducing a fine-grained hierarchical approach to tackle adversarial attacks by leveraging patterns extracted from the basic and elementary structure of the tested software.

To this end, we start by investigating the robustness of DL-SSMC, **D**eep **L**earning-based **S**ingle **S**hot **M**alware **C**lassification approach, for accurate IoT malware detection and classification. DL-SSMC utilizes machine learning techniques for IoT malware detection and classification, taking feature representation as an input, and outputs the classification corresponding to the input, we refer to this process as "single shot", as it requires querying the system once, and no decisions are taken outside of the machine learning model itself. Then, we examine the approach against AEs generated by Graph Embedding and Augmentation (GEA) [1] and Sub-Graph Embedding and Augmentation (SGEA) [20]. The GEA and SGEA are graph-based AEs generation approaches that are proposed to bypass CFG-based malware detection systems.

To cope with adversaries and to minimize their effects, we propose DL-FHMC, **F**ine-grained **H**ierarchical Learning for **M**alware **C**lassification, for detecting and classifying malware samples by operating on a fine-grained level of structures and patterns extracted from the malicious software. DL-FHMC utilizes the shared behavioral structures of the malicious IoT software from the same family to create sub-graph signatures representing the execution flows of the malicious behavior. The extracted signatures are then used to distinguish benign and AEs with high accuracy. Our experiments show the effectiveness of our proposed approach in detecting malware samples as well as a high-degree of robustness against a variety of adversarial attacks.

Summary of contributions. Our contributions are as follows:

- Investigate the robustness of DL-SSMC for IoT malware detection and classification under GEA and SGEA adversarial configurations. Through comprehensive experiments, we show the effectiveness of GEA and SGEA in producing successful AEs that can fool the machine learning-based malware detection system.
- Propose DL-FHMC, fine-grained hierarchical learning for malware classification, that extracts potential malicious behavioral patterns of IoT malicious families. We extracted 30,000 behavioral patterns from three IoT malicious families.
- Investigate the robustness of DL-FHMC under adversarial configurations. DL-FHMC operates by investigating the malicious subgraphs within the IoT malware, using subgraph mining and pattern recognition to detect suspicious and malicious behaviors within the tested samples, mitigating the effects of AEs and detecting up to 100% of malicious AEs.

Organization. This work is organized as follows: In §2, a brief background is provided. We discuss the threat model and adversarial capabilities in §3. An overview of the dataset, data representation, and used learning algorithms are highlighted in §4. In §5, we evaluate the proposed adversarial methods on traditional deep learning techniques. Then, we propose a fine-grained hierarchical learning technique for suspicious behavior detection in §6. In §7, we discuss the proposed adversarial methods and suspicious behavior detection approach. Related work has been discussed in §8. Finally, we conclude our work in §9.

2 Preliminaries

We investigate the effect of incorporating adversarial learning techniques into CFG-based deep learning IoT malware detection systems in an attempt to understand the robustness of such models against adversarial learning attacks as a result of AEs. This will in turn provide insights toward proposing a robust IoT adversarial software detection system. In the following, we provide a brief background related to malware analysis, and incorporating graph theory into analyzing malicious behavior.

2.1 Malware Analysis

Malware analysis is used for understanding malware functionality, capabilities, behavior, and intent. The results of the analysis are used to build detectors and design defenses to protect against future malware campaigns, while the analysis falls into one of two types, static or dynamic. Static analysis examine the binary without execution. Given the malicious nature such binaries, static analysis is utilized as a precursor to dynamic analysis. The malware binary can then be executed in a sandboxed environment with a much-reduced focus to observe the patterns, like the behavioral artifacts in which is called the dynamic analysis.

Static Analysis. Static analysis approaches employ techniques to extract indicators to determine whether a binary is malicious or benign [21]. The various analysis modalities, such as instructions, basic blocks, functions *etc.*, hint at the possible execution pattern of the software. For example, the traces of user name and password list, along with shell-based login attempts, implies possible usage of dictionary attacks being launched by the software. These inferential results are drawn from static analysis to enable the analysts to emphasize and predict specific patterns. Additionally, traces can also be used by analysts to address issues during dynamic analysis, *e.g.,* virtual machine obfuscation, ptrace obfuscation *etc.*. Traces and components of the software are often extracted using a reverse engineering process of the software binaries that allow the understanding of its composition, architecture, and design. Analysts perform reverse engineering of software binaries using several available off-the-shelf tools. The reverse engineering process also generates artifacts to be subject of analysis including high-level representation of the binaries such as the CFGs and Data Flow Graphs (DFGs). The CFG of a program is the graphical representation of the flow of control during the execution of that program. While the DFG represents the system events to understand the possible execution of the system behaviors. It explains the flow of the data that passes from one node to another. Although static analysis is quite powerful and popular, it sometimes falls short of achieving its end goal of providing in-depth insights to the software due to multiple evasion techniques. For example, malware developers use evasion techniques to the analysis of their produced malware. The evasion techniques include packing (UPX [22]), obfuscation (function-, string- obfuscation), *etc.*.

Dynamic Analysis. Unlike static analysis, dynamic analysis executes the application in a simulated and monitored environment to observe its behavior and understand its functionality [23]. This approach aims to capture different behavioral patterns of software. For example, dynamic analysis helps to unravel the program's network patterns, such as communication with the Command and Control (C&C) server. Since the malicious nature of software can affect the status of the machine it is executed on, the following observations are made: 1) comparing the system state before and after the execution of the application, or 2) monitoring the application's actions during the execution. Similar to static analysis, dynamic analysis can be evaded by software developers by adopting means that prevent their software from getting dynamically-analyzed. For example, malware developers often employ conditions that crash the software once encountering virtual machines, debugging tools, or network monitoring tools.

2.2 Graph Analysis

Graph Analysis. The CFG is a graph representation of the program which shows all paths that can be reached during the execution, as shown in Figure 1. In a CFG, the set of nodes means the basic blocks where each block is a straight-line instruction without any *jump* or *jump target*, while the set of directed edges corresponds to the *jump* which traverses from the block to the other block at the branch (*if*), loop (*while, for*), and the end of the function (*return*). Once the first instruction of the basic block is executed, the rest of the instructions in the same block are necessarily executed unless terminated by external interference. In general, CFG is used for the structural analysis of the program. For example, from the perspective of optimization, the CFG is used to analyze the reachability of each block. By constructing the CFG and evaluating the reachability, the flaws of the program (infinite loop or unreachable codes) can be found and addressed.

CFG-based Analysis. In graph theory, there are various concepts that express the characteristics of a graph. Given $G = (V, E)$, for example, the number of vertices ($|V|$) means the order of G, while the number of edges ($|E|$) corresponds to the size of G. The density of the graph can be defined as $D = |E|/(|V| * (|V| - 1))$ for directed simple graph, which means the ratio of the number of edges in G to the maximal number of edges in the complete graph. The centrality is measured for each node $v \in V$, which shows how important a specific node is. In detail, there are several different kinds of centrality, such as closeness centrality, betweenness centrality, Eigenvector centrality, *etc.*.

These indicators (and further concepts not described above) can be considered the features of the graph G. Moreover, the combination of those metrics can be a more deterministic characteristic of the graph. Considering that a CFG is a kind of graph, it is true that each binary has not only its unique graph representation but also the associated values, such as the order, size, and density of CFG, and centrality for each vertex in CFG. On the other hand, the graph-based analysis can provide the possibility for identifying the malware. Because it is highly likely that the binaries in the same "family" share the structural similarity (even if there is a little difference), the CFG-based features can be combined with the state-of-the-art machine learning technique to determine whether a given binary is malicious or not.

```
#include <stdio.h>
void main(){
    int a = 0;
    do
    {
      a++;
    }while(a < 10);
}
```

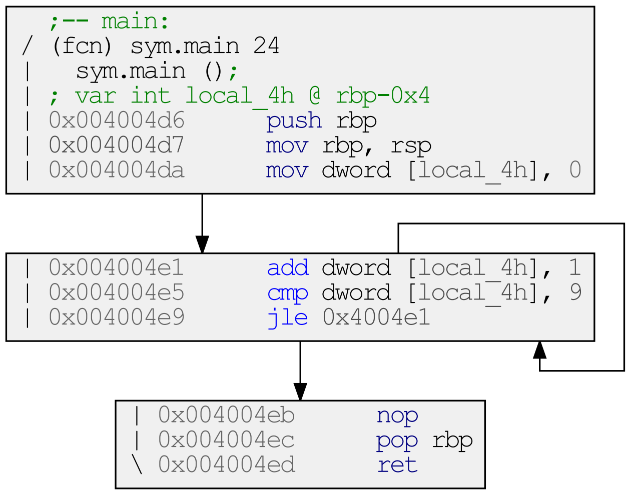
Listing 1: C script of an example of original sample

```
#include <stdio.h>
void main(){
    int x = 0;
    int s = 0;
    if (x!=0){
        s++;
    }
}
```

Listing 2: C script of an example of targeted sample

```
#include <stdio.h>
void main(){
    /*set a condition variable*/
    int cond=1;
    if (cond==1){
        /*script of original sample*/
        /*this section will be executed*/
        int a = 0;
        do{
            a++;
        }while(a<10);
    }
    else{
        /*script of target sample*/
        /*this section will not be executed*/
        int x = 0;
        int s = 0;
        if (x!=0){
            s++;
        }
    }
}
```

Listing 3: C script of combining original and selected samples. Note that a condition variable is used to enable the desired functionality. Additionally, the script of the original and selected samples are preserved within the generated sample.

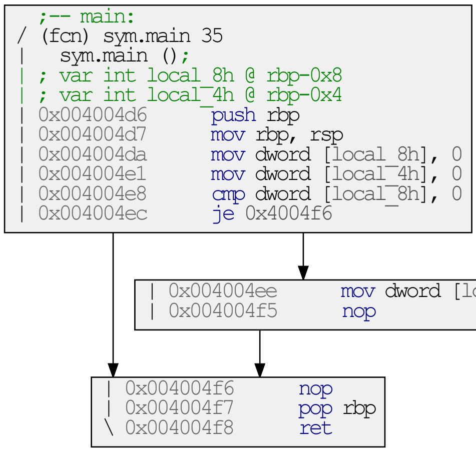


```
  ;-- main:
/ (fcn) sym.main 24
|   sym.main ();
| ; var int local_4h @ rbp-0x4
| 0x004004d6      push rbp
| 0x004004d7      mov rbp, rsp
| 0x004004da      mov dword [local_4h], 0

| 0x004004e1      add dword [local_4h], 1
| 0x004004e5      cmp dword [local_4h], 9
| 0x004004e9      jle 0x4004e1

| 0x004004eb      nop
| 0x004004ec      pop rbp
\ 0x004004ed      ret
```

Fig. 1: The generated CFG for the original sample and used for extracting graph-based features (graph size, centralities, etc.) for graph/program classification and malware detection.

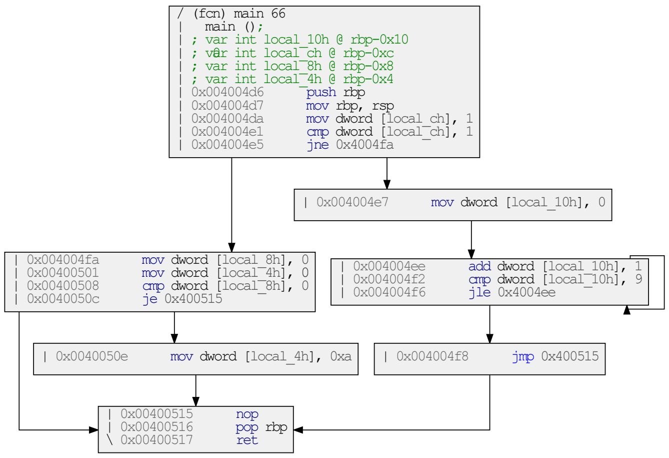


```
  ;-- main:
/ (fcn) sym.main 35
|   sym.main ();
| ; var int local_8h @ rbp-0x8
| ; var int local_4h @ rbp-0x4
| 0x004004d6      push rbp
| 0x004004d7      mov rbp, rsp
| 0x004004da      mov dword [local_8h], 0
| 0x004004e1      mov dword [local_4h], 0
| 0x004004e8      cmp dword [local_8h], 0
| 0x004004ec      je 0x4004f6

| 0x004004ee      mov dword [local_4h], 0xa
| 0x004004f5      nop

| 0x004004f6      nop
| 0x004004f7      pop rbp
\ 0x004004f8      ret
```

Fig. 2: The CFG for the selected target sample generated and used for extracting graph-based features (graph size, centralities, etc.) for graph/program classification and malware detection.



```
/ (fcn) main 66
|   main ();
| ; var int local_10h @ rbp-0x10
| ; var int local_ch @ rbp-0xc
| ; var int local_8h @ rbp-0x8
| ; var int local_4h @ rbp-0x4
| 0x004004d6      push rbp
| 0x004004d7      mov rbp, rsp
| 0x004004da      mov dword [local_ch], 1
| 0x004004e1      cmp dword [local_ch], 1
| 0x004004e5      jne 0x4004fa

| 0x004004e7      mov dword [local_10h], 0

| 0x004004fa      mov dword [local_8h], 0
| 0x00400501      mov dword [local_4h], 0
| 0x00400508      cmp dword [local_8h], 0
| 0x0040050c      je 0x400515

| 0x004004ee      add dword [local_10h], 1
| 0x004004f2      cmp dword [local_10h], 9
| 0x004004f6      jle 0x4004ee

| 0x0040050e      mov dword [local_4h], 0xa

| 0x004004f8      jmp 0x400515

| 0x00400515      nop
| 0x00400516      pop rbp
\ 0x00400517      ret
```

Fig. 3: The generated adversarial graph using GEA approach. Note that this graph is obtained logically by embedding the graph in Fig. 2 into the graph in Fig. 1, although indirectly done by injecting the code listings as highlighted in Listings 1, 2, and 3.

3 Threat Model

The rapid reliance on machine learning methods in various applications has raised several security and privacy concerns, especially in security-sensitive applications. It has become crucial to understand and assess the robustness of machine learning techniques to several adversarial settings. These adversarial settings include AEs, where an adversary intends to fool or misguide the classification model with malicious inputs that are generated by applying a minimal perturbation to the original sample [18]. These modifications misclassify the samples of the model from benign to malware and vice versa and even misclassify the malware classes to another class. Such adversarial attacks can be launched under different adversarial capabilities that allow for either black-box and white-box attacks. In a white-box attack, the adversary has full knowledge of the inner networking paradigm of the model, while in a black-box attack, the adversary has only access to the model via an oracle and observes only the output of the model.

The literature on AEs and their effects includes numerous studies where the perturbation is applied to image pixels [15], [18], [24]. Unlike image AEs, the generated AEs from the IoT software must preserve the original sample's functionality and practicality in order to function properly. Adversarial machine learning can be derived from two perspectives: targeted and non-targeted attacks.

Targeted attacks. The focus of this attack is to generate AE x' that forces the classifier f to misclassify into a specific target class t. For instance, the adversary generates a set of malicious IoT software samples, which are classified as benign. That is: $x' : [f(x') = t] \wedge [\Delta(x, x') \leq \epsilon]$, where $f(.)$ represents the classifier's output, $\Delta(x, x')$ denotes the difference between x and the crafted AE x', whereas ϵ is a distortion threshold.

Non-targeted attacks. The focus of non-targeted attack is to generate an AE that forces the classifier f to misclassify to any class other than the original class $f(x)$, where x is the original input. That is: $x' : [f(x') \neq f(x)] \wedge [\Delta(x, x') \leq \epsilon]$, where $f(.)$ shows the classifier's output, $\Delta(x, x')$ represents the difference between x and x', and ϵ is the distortion threshold.

In this study, we generate AEs from the IoT software based on code-level manipulation using GEA [1] and SGEA [20]. In the following, we discuss each attack briefly.

3.1 Graph Embedding and Augmentation (GEA)

GEA generates realistic AEs, where the functionality and practicality of the original binary are maintained. The key idea of GEA is combining an original CFG with a targeted CFG. In the following, we briefly describe GEA using an example.

Practical Implementation. Assume an original sample (software) (x_{org}) and a selected target sample (x_{sel}), GEA combines x_{org} with a x_{sel} while preserving the functionality and practicality of x_{org}. Listing 1 shows the sample script of x_{org}, and Listing 2 shows a sample script of x_{sel}. The GEA process combines the two scripts while ensuring that x_{sel} does not affect the process and functionality of x_{org}. The generated AE in Listing 3 shows the script after the combination process. Note that the condition is set to execute only the functionality related to x_{org} and preventing the processes of x_{sel} from being executed. Prior to generating the CFG for these algorithms, we compile the code using the GNU Compiler Collection (GCC) command. Afterward, Radare2 [25] is used to extract the CFG from the binaries.

Figure 1 and Figure 2 show the generated CFGs for both x_{org} and x_{sel}, respectively. As shown in Figure 3, the combined CFG consists of the two scripts sharing the same entry and exit nodes. Therefore, the GEA approach adds modifications to the CFG for generating the AE. Given the nature of the extracted features, the applied changes on the CFG are reflected upon the features, regardless of the effects on the functionality and executability of the original sample. Following the adopted approach in [1], we select three different-sized graphs from benign samples as x_{sel}. The selected graphs vary in size, where the size is the number of nodes in the graph. To generate AEs, we selected a graph and connected it with all malicious samples.

3.2 Sub-GEA (SGEA)

While GEA combines an original CFG to a selected CFG of the IoT samples to misclassify the machine learning model, the SGEA approach aims to reduce the injection size and achieve the adversarial objectives with minimal perturbation. More specifically, it uses deep discriminative subgraph patterns extracted from the CFGs of each class using a correspondence-based quality criterion (CORK) algorithm, which defines a submodular quality criterion that ensures a solution close to the optimal solution [26]. This is done by using subgraphs that appear more frequently in one class than others, to fool the machine learning model in predicting

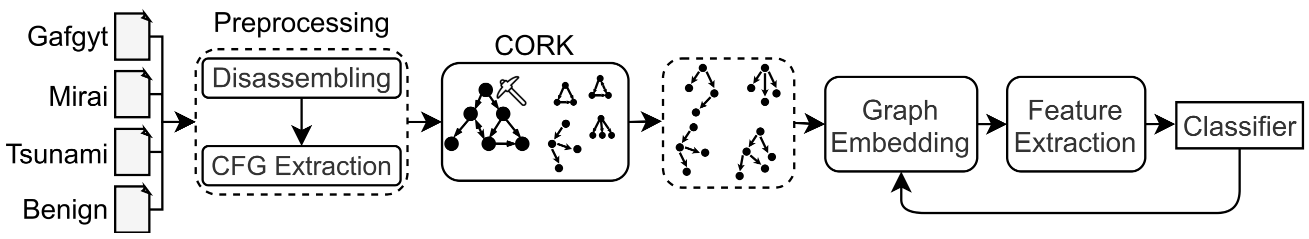


Fig. 4: SGEA pattern extraction and AE generation process. SGEA uses CORK to extract discriminative subgraphs from each class. Then, the extracted subgraphs are embedded to generate the AEs. The process is terminated and the AE is returned upon successfully misclassifying the model.

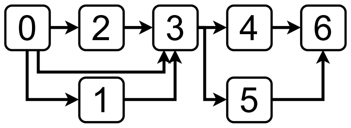


Fig. 5: Sample of extracted discriminative subgraph from Gafgyt malicious family. Here, the graph size is 7, and the labels are arbitrary. Ideally, connecting this subgraph to a sample should lead the model to misclassify the sample into Gafgyt.

```
#include<stdio.h>
void  main(){
      int GEAVar1 = 0; // block 0
      if (GEAVar1 == 1){ // block 1
           GEAVar1 += 1;
      }
      else if (GEAVar1 == 2){ // block 2
           GEAVar1 += 2;
      }
      int GEAVar2 = 0; // block 3
      if (GEAVar2 == 0){ // block 4
           GEAVar2 += 1;
      }
      else{ // block 5
           GEAVar2 += 2;
      }
      int GEAVar3 = 0; // block 6
}
```

Listing 4: C script of an example Gafgyt extracted subgraph. Each block is represented as a node in the generated CFG. Appending this code to the source code of a sample will lead to producing the subgraph shown in Figure 5.

that class (*e.g.,* when launching a targeted attack). Let D denotes the CFGs of the training samples, $D = \{G_i\}_{i=1}^{n}$ and class labels $C = \{c_i\}_{i=1}^{n}$ where $c_i \in \{+1, -1\}$ is the class label of graph G_i. Also let D^+ and D^- denote the set of graphs in the corresponding classes. For a multi-class dataset, we run the CORK algorithm once for each class where all the graph that belong to the same class are included in D^+ and the other graphs are in D^-. A graph G_i supports another graph S if S is a subgraph of G_i. Let $D_S = \{G_i | S \subseteq G_i \; \forall \; G_i \in D\}$ denote the supporting graphs of a subgraph S. Moreover, let D_S^+ and D_S^-, denote the supporting graphs of the subgraph in the positive and negative graphs, respectively. CORK defines a submodular quality criterion, q, for a subgraph based on the set of supporting graphs ('hits') and non-supporting graphs ('misses') in the two classes and is calculated as follows: $q(G_s) = -(|D_S^{+\sim}| * |D_S^{-\sim}| + |D_S^+| * |D_S^-|)$. The best quality score is achieved when a subgraph appears in all graphs of one class and not once in the graphs of the other classes. Pruning strategies, as used in the quality criterion of CORK, are integrated into the gSpan algorithm [27] to directly mine discriminative subgraphs. Once the set of discriminative subgraphs are mined, we employ gSpan, a graph-based substructure mining pattern for mining frequent subgraphs of size five nodes or higher.

TABLE 1: Distribution of IoT samples across the classes. We split the dataset into 80% training and 20% testing, with an overall 10,091 IoT samples (7,091 IoT malware and 3,000 benign).

Class		# of Samples			% of Samples
		# Train	# Test	# Total	
Benign		2,400	600	3,000	29.72
Malicious	Gafgyt	2,400	600	3,000	29.72
	Mirai	2,400	600	3,000	29.72
	Tsunami	872	219	1091	10.84
Overall		8,072	2,019	10,091	100

Practical Implementation. SGEA combines x_{org} with the selected discriminative subgraph (x_{sel}). For example, Figure 5 shows the discriminative subgraph extracted from the Gafgyt class and listing 4 shows the equivalent C script to generate that subgraph, which can then be combined with the x_{org} to generate an AE. Figure 4 shows the overview of patterns extraction and the process of generating AEs in the SGEA approach. While GEA modifies the CFG by connecting the selected graph with the original sample, SGEA connects a carefully generated subgraph with the original sample to generate AE, reducing the injected graph size. To generate the subgraph, we extracted the discriminative subgraph patterns from each class, with a size of five nodes or higher. Then, in order to reduce the graph size needed to be embedded, we connect x_{org} with the subgraph with minimum size. If the generated AE misclassifies, the process succeeds, and the AE will be returned; else, we select the next subgraph in ascending order regarding the number of nodes in the subgraph. In case none of the subgraphs cause misclassification, the original sample will be returned as the process failed.

Constructing an AE. As shown in Figure 4, we extract a set of subgraphs from the targeted class. Then, we combine the original sample with the smallest extracted subgraph of the targeted class regarding the number of nodes. If the generated CFG fails to misclassify the model, another subgraph is selected in ascending order with respect to the number of nodes in the set of generated subgraphs and combined with the same original graph to generate another CFG. This process is repeated until a subgraph successfully misclassifies the model. If no existing subgraph from the set of targeted subgraphs causes misclassification, the original sample is returned, hence the process failed in generating AE. In this study, we consider AEs that misclassify malware to benign, as such AEs have huge risk on the users, and render the malware detector systems useless.

TABLE 2: The distribution of extracted features. 23 algorithmic features are extracted from the CFGs. These features are categorized into seven groups, including number of nodes and edges, density, shortest path, and centralities. When possible, the minimum, maximum, median, mean, and standard deviation values are extracted, as in the shortest path group.

Feature category	# of features
Betweenness centrality	5
Closeness centrality	5
Degree centrality	5
Shortest path	5
Density	1
# of Edges	1
# of Nodes	1
Total	23

4 Data Representation & Learning

In this section, we discuss the utilized dataset, dataset representation, and learning algorithms, including the experimental setup for DL-SSMC and DL-FHMC.

4.1 Dataset

In this work, we collected binaries of two categories, IoT malicious and benign samples. The malicious samples are collected from CyberIOCs [28], VirusTotal [29], and VirusShare [30] in the period of January 2018 to late January of 2021, with a total of 7,091 samples that belong to three malware families. Additionally, we assembled a dataset of 3,000 benign IoT samples compiled from the source files on GitHub [31].

Ground Truth Class. The benign and malicious samples in our dataset were validated using the *VirusTotal* [29]. We uploaded the samples on VirusTotal and gathered the scan results corresponding to each sample. We then used AVClass [32] to classify the malicious samples into their corresponding families. We summarize the dataset in table 1.

4.2 Data Representation

Samples of the IoT benign (3,000 sample) and IoT malware categories (7,091 samples) were reverse-engineered using *Radare2* [25], a reverse engineering framework that provides various analysis capabilities, for obtaining the samples' corresponding CFGs. Using the samples' CFGs extracted by *Radare2*, we represent the CFG using the graph-theoretic features proposed by Alasmary *et al.* [33]. In particular, we extracted 23 different algorithmic features categorized into seven groups. Table 2 shows the feature category and the number of features in each category. Except for the number of edges, nodes, and the density of the graph, five features were extracted from each feature category, including the minimum, maximum, median, mean, and standard deviation values for the observed parameters. To this end, each IoT software is represented as a vector of size 1×23 representing the corresponding CFG-based algorithmic features.

4.3 Learning Algorithms

Toward IoT malware detection and classification, we utilize different machine and deep learning algorithms for pattern learning and deep feature extraction. In the following, we briefly describe each learning algorithm.

Random Forest (RF). RF consists of N decision trees, each decision tree is trained on a collection of random features, and finds the non-linear relationships between the features and the output decision. The final prediction of RF classifier with N decision trees is determined by a majority vote over the predictions or by averaging the prediction of all trees, determined as follows:

$$f_{RF} = \frac{1}{N}\sum_{n=1}^{N} f_n(X_s^{'}),$$

where, for a randomly selected feature set, $(X_.^{'} \subset X_.)$, f_n is the n^{th} tree's prediction and $X_s^{'}$ is the segment's s vector.

Deep Neural Networks (DNN). DNN is an artificial neural network with neurons of each layer are fully connected to the neurons of the next layer. It consists of multiple hidden layers between the input and output layers. Given a feature vector X of length q and target y, the DNN-based classifier learns a function $f(.) : R^q \to R^o$, where q is the input's dimension and o is the output's dimension. With multiple hidden layers, the dimension of the output of every hidden layer decreases with transformation. Each neuron in the hidden layer transforms the values of the preceding layer using linearly weighted summation, $w_1 + w_2 + w_3 + ...w_q$, which passes through a ReLU activation function $(y(x) = max(x, 0))$. The output of the hidden layers is then fed to the output layer, and passed to a softmax activation function h, defined as $h(x) = \frac{1}{1+e^{-x}}$, outputting the prediction of the classifier.

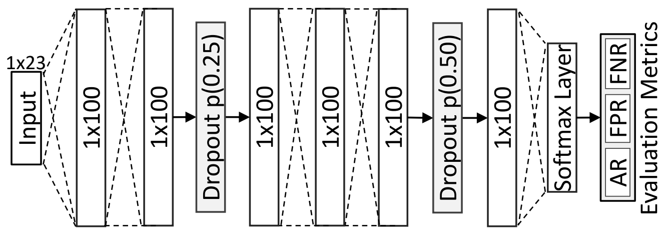


Fig. 6: The internal architectural of the DNN used for the detection and classification tasks. The design consists of six fully connected layers, with dropout operations and softmax activation function.

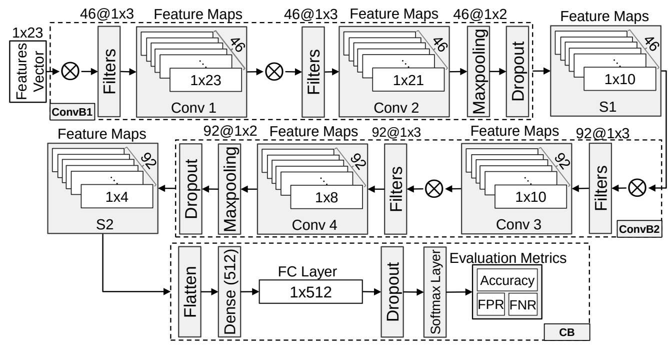


Fig. 7: Internal design of the CNN architecture used for detection and classification task. Notice that 46@1x3, for example, refers to applying 46 filters each of size 1x3 on the input data. The design consists of four convolutional layers with maxpooling and dropout operations. Then, a dense layer of size 512 is used with a softmax activation function to output the model's prediction.

Figure 6 illustrates the DNN-based model utilized for training the IoT malware detector and classification system. The architecture of the DNN-based model consists of two consecutive and fully connected dense layers of size 1×100 connected to the input vector, followed by a dropout operation with a probability of 0.25. The output of the dropout function is fed to fully connected with

	B	G	M	T
B	0.995	0.002	0.002	0.002
G	0.007	0.977	0.015	0.002
M	0.015	0.008	0.977	0.000
T	0.009	0.018	0.041	0.931

(a) RF

	B	G	M	T
B	0.960	0.005	0.025	0.010
G	0.007	0.978	0.013	0.002
M	0.017	0.037	0.937	0.010
T	0.023	0.023	0.023	0.931

(b) DNN

	B	G	M	T
B	0.963	0.005	0.023	0.008
G	0.010	0.973	0.015	0.002
M	0.013	0.018	0.967	0.002
T	0.041	0.018	0.041	0.899

(c) CNN

Fig. 8: Confusion matrices of IoT malware classification systems. Here, each row represents the actual class, whereas, columns represents the predicted labels. Labels are Benign (B), Gafgyt (G), Mirai (M), and Tsunami (T).

	B	G	M	T
B	0.000	0.000	0.000	0.000
G	0.792	0.183	0.003	0.022
M	0.326	0.002	0.671	0.002
T	1.000	0.000	0.000	0.000

(a) RF-Small

	B	G	M	T
B	0.000	0.000	0.000	0.000
G	1.000	0.000	0.000	0.000
M	0.992	0.000	0.007	0.001
T	1.000	0.000	0.000	0.000

(b) RF-Median

	B	G	M	T
B	0.000	0.000	0.000	0.000
G	1.000	0.000	0.000	0.000
M	1.000	0.000	0.000	0.000
T	1.000	0.000	0.000	0.000

(c) RF-Large

	B	G	M	T
B	0.000	0.000	0.000	0.000
G	0.880	0.100	0.000	0.020
M	0.492	0.028	0.472	0.007
T	0.972	0.000	0.009	0.018

(d) DNN-Small

	B	G	M	T
B	0.000	0.000	0.000	0.000
G	1.000	0.000	0.000	0.000
M	0.876	0.093	0.030	0.000
T	1.000	0.000	0.000	0.000

(e) DNN-Median

	B	G	M	T
B	0.000	0.000	0.000	0.000
G	1.000	0.000	0.000	0.000
M	1.000	0.000	0.000	0.000
T	1.000	0.000	0.000	0.000

(f) DNN-Large

	B	G	M	T
B	0.000	0.000	0.000	0.000
G	0.078	0.608	0.017	0.297
M	0.135	0.000	0.836	0.028
T	0.885	0.009	0.037	0.069

(g) CNN-Small

	B	G	M	T
B	0.000	0.000	0.000	0.000
G	0.560	0.438	0.002	0.000
M	0.873	0.020	0.107	0.010
T	1.000	0.000	0.000	0.000

(h) CNN-Median

	B	G	M	T
B	0.000	0.000	0.000	0.000
G	1.000	0.000	0.000	0.000
M	1.000	0.000	0.000	0.000
T	1.000	0.000	0.000	0.000

(i) CNN-Large

Fig. 9: GEA: Confusion matrices of IoT malware classification systems. Here, each row represents the actual class, whereas, columns represents the predicted labels. Labels are Benign (B), Gafgyt (G), Mirai (M), and Tsunami (T).

	B	G	M	T
B	0.000	0.000	0.000	0.000
G	1.000	0.000	0.000	0.000
M	0.995	0.000	0.005	0.000
T	1.000	0.000	0.000	0.000

(a) RF

	B	G	M	T
B	0.000	0.000	0.000	0.000
G	0.985	0.000	0.015	0.000
M	0.657	0.139	0.194	0.010
T	0.930	0.000	0.070	0.000

(b) DNN

	B	G	M	T
B	0.000	0.000	0.000	0.000
G	0.602	0.000	0.348	0.005
M	0.736	0.000	0.219	0.045
T	0.915	0.015	0.070	0.000

(c) CNN

Fig. 10: SGEA: Confusion matrices of IoT malware classification systems. Here, each row represents the actual class, whereas, columns represents the predicted labels. Labels are Benign (B), Gafgyt (G), Mirai (M), and Tsunami (T).

another two fully connected dense layers of size 1×100, followed by a dropout operation with a probability of 0.5. The output is then fed to the softmax layer. This design enables the extraction of deep feature representations and patterns from the feature vectors, and therefore, finding discriminative characteristics for detection and classification processes.

Convolutional Neural Network (CNN). The basic unit of the CNN network is a convolution layer, which consists of several filters convolving over the input to generate feature maps. Once a feature vector is fed into a convolutional layer, it becomes abstracted to a feature map, with the shape of (feature map height) $\times$ (feature map width) $\times$ (feature map depth), with two attributes: 1) convolutional kernels defined by a width and a height (hyper-parameters), 2) the depth of the convolution filter, which is equal to the depth of the input vector representation feature map. The CNN-based model constitutes of three blocks, two feature extraction layers, along the classification layer. Figure 7 illustrates the CNN-based model architecture used in this study.

Even though, from a machine learning prospective, RF model seems the most suitable given the data representation and structure, and the limited number of features (*i.e.,* 23), we also used DNN and CNN-based approaches as they show their capabilities in multiple applications for automatically producing high quality encoding of the features towards a highly accurate classification. We note that alongside the aforementioned approaches, we also evaluate the performance of Logistic Regression- and Support Vector Machine-based architectures, however, due to their low performance, the results were not included in the evaluation.

4.4 Experimental Setup

We consider both malware detection and classification. Malware classification refers to identifying a malware family a sample, while the detection is simply indicating whether a sample is malicious or benign. Therefore, the detection task can be viewed as a binary classification task. The classification task aims to detect and identify the malicious behavior origin (*i.e.,* family).

Training Process. We trained the model architectures using 100 epochs with a batch size of 32. RF uses 100 decision trees, with no specified maximum length. For DNN and CNN, we used Rectified Linear Units (ReLUs) as the activation function, with softmax activation function at the classification layer. For regularization, we use dropout to prevent over-fitting and allow propagation of

TABLE 3: Evaluation (%) of the IoT malware detection systems on normal samples (*i.e.,* non-adversarial).

Architecture	Accuracy	F-1	FNR	FPR
RF	98.90	99.22	1.19	0.83
DNN	97.42	98.16	1.69	4.67
CNN	98.31	98.80	1.12	3.00

TABLE 4: Evaluation (%) of the IoT malware classification systems on normal samples (*i.e.,* non-adversarial).

Architecture	Accuracy	F-1
RF	97.71	97.71
DNN	95.53	95.52
CNN	96.03	96.02

robust and distinct features through the model layers. Note that the value 0.25 is widely used within the machine learning community, as 75% of the neurons are considered for feature propagation between layers [34]. This provides better accuracy, and mitigate the noise caused by possible bias within the dataset.

Evaluation Metrics. We report the performance of the trained models using the following metrics: 1) The accuracy of the model, computed as the ratio of the correctly labeled samples (CLS) overall test samples ($|D|$), defined as: $CLS \div |D|$. 2) False Positive Rate (FPR), which is the number of incorrectly labeled benign samples (ILB) over the total number of benign samples ($|D_b|$), computed as $ILB \div |D_b|$. 3) True Positive Rate (FNR), represented as the correctly labeled malicious samples (CLM) divided by the total number of malicious samples ($|D_m|$), $CLM \div |D_m|$. 4) The F-1 score, defined as: F-1 $=$ 2TP/(2TP + FP + FN), where TP: the number of malicious samples correctly classified, FP: the number of benign samples incorrectly classified, FN: the number of malware samples incorrectly classified. 5) Misclassification rate, defined as the ratio of the incorrectly labeled samples over all the samples in the test dataset (*i.e.,* $1-$ accuracy).

We also report the confusion matrix when required. The rows represent the actual classes and the columns are the predicted labels. The value at a location (x,y) represents the portion of the samples of class x classified as y.

5 DL-SSMC: Design and Evaluation

This section presents DL-SSMC, **D**eep **L**earning-based **S**ingle **S**hot **M**alware **C**lassification approach. We describe the design and methods for DL-SSMC in §5.1 and present the evaluation in §5.2.

5.1 DL-SSMC: System Design

The implementation of DL-SSMC incorporates machine and deep learning models trained using the extracted CFG-based algorithmic features for malware detection and classification tasks. In this approach, we follow the traditional learning approach. The input (X) to the model is a one-dimensional (1D) vector of size 1×23 representing the extracted features. Using a Softmax activation function, the model outputs whether the software is benign or malicious, alongside the predicted family in the classification task, *i.e.,* Gafgyt, Mirai, or Tsunami. To this end, we utilize the aforementioned architectures (subsection 4.3) to train the models for both detection and classification tasks.

TABLE 5: GEA: Malware to benign misclassification rate (%) over IoT detection systems.

Architecture	Small	Median	Large
	10	23	1,075
RF	73.88	99.85	100
DNN	37.54	90.04	100
CNN	18.84	64.78	100

TABLE 6: GEA: IoT classification systems misclassification rates (%). The CNN-based model perform the best, robustness-wise, under the small and median GEA graph embedding attacks.

Architecture	Size	Gafgyt	Mirai	Tsunami
RF	Small	81.66	32.88	100
	Median	100	99.33	100
	Large	100	100	100
DNN	Small	90.00	52.75	98.16
	Median	100	96.99	100
	Large	100	100	100
CNN	Small	39.16	16.36	93.11
	Median	56.16	89.31	100
	Large	100	100	100

5.2 DL-SSMC: Evaluation and Results

5.2.1 DL-SSMC: Baseline Performance

DL-SSMC: Detection Task. We design two-class detection DL-SSMC that distinguish IoT malware from the IoT benign applications. The model is trained over 23 CFG-based graph-theoretic features categorized into seven groups. The models achieve an accuracy rate of 98.90%, 97.42%, and 98.31% with an F-1 score of 99.22%, 98.16%, and 98.80% for RF, DNN, and CNN, respectively. Table 3 shows the evaluation of each trained model. Notice that the RF-based model holds the highest performance, followed by the CNN model.

DL-SSMC: Classification Task. In addition to detecting the IoT malicious samples, we also design a four-class classification DL-SSMC. The classification task aims to evaluate DL-SSMC for classifying the malicious samples into their corresponding families. We achieved accuracy rates of 97.71%, 95.53%, and 96.03% for RF-, DNN-, and CNN-based models, respectively, as shown in Table 4. We also provide the confusion matrices (represented as a percentage of samples) in Figure 8. Here, each row represents samples of an individual class, while the columns represent the predicted family.

5.2.2 DL-SSMC: Robustness Assessment against GEA

We investigate the robustness of DL-SSMC against AEs generated using GEA. In particular, we explore the impact of the size of the graph and discuss the fundamental overhead of using GEA. Note that all generated samples maintain the practicality and functionality of the original code. From the benign software, we selected three graphs as x_{sel}, the selected samples have small, median, and large sizes across the dataset. We then connected each of the graphs with every graph in the malware dataset.

Robustness of Detection Models. The results of DL-SSMC performance against AEs generated by GEA are shown in Table 5 and Figure 9. Intuitively, a key finding is the impact of graph size on the misclassification rate, since the increase in the graph size, *i.e.,* the included number of nodes, results in a higher misclassification rate. The main reason for the misclassification is the injection of benign-like patterns that introduce noise that distort the existing malicious patterns observed by the learning algorithm. Embedding larger graphs, for instance, introduces higher distortion to the feature space, considering the extracted 23 algorithmic

TABLE 7: SGEA: Malware to benign IoT malware detection system evaluation. Here, MR: misclassification rate, AVG. Size: the average subgraph size required to achieve misclassification.

Architecture	MR (%)	AVG. Size
RF	99.17	6.28
DNN	90.21	6.83
CNN	41.79	7.15

TABLE 8: SGEA: Misclassification rate (%) over IoT classification systems. MR: misclassification rates.

Architecture	Gafgyt	Mirai	Tsunami
RF	100	99.50	100
DNN	100	80.59	100
CNN	100	78.10	100

features. This distortion affects the existing patterns learnt by the machine/deep learning model, and therefore causes higher misclassification. In general, GEA achieves a misclassification rate of 100% by embedding a large graph, while achieving a low as 18.84% misclassification rate by embedding a small graph. Notice that while the RF-based model provides the best clean baseline performance, the CNN-based architecture shows the best robustness against embedding small and medium graphs.

Robustness of Classification Models. Table 6 shows the misclassification rates over the IoT malware classification task. Here, GEA achieves a misclassification rate of 100% from all malicious families using large graph embedding. Similarly, Tsunami is more likely to be misclassified to benign, as using median and large graph embedding misclassify all Tsunami malware to benign as shown in Figure 9. Similar to the IoT malware detection system, the misclassification rates increase with the increase in the number of nodes for the injected graph, and the CNN-based model shows better robustness (*i.e.,* lower misclassification rate) against graph embedding, in comparison to its counterparts.

5.2.3 DL-SSMC: Robustness Assessment against SGEA

While GEA achieves a high misclassification rate, it comes with a computational cost and increased binary size that accommodates the combination of two samples into one. These costs are reduced by using SGEA, in which, the size of injection is reduced by carefully selecting a subgraph that achieves the adversarial objective.

Robustness of Detection Models. Table 7 show the results of SGEA against DL-SSMC detection models. SGEA achieves above 90% malware to benign misclassification rate against RF- and DNN-based models with less than 7 nodes subgraph embedding, outperforming the GEA approach with an average subgraph size of 6.28 and 6.83, respectively. However, for CNN, the misclassification rate is noticeably lower, 41.79%, as the CNN-based model shows better robustness against GEA and SGEA.

Robustness of Classification Models. Table 8 and Figure 10 show the performance of the DL-SSMC classification models against the SGEA approach. For instance, the SGEA approach successfully misclassifies all Gafgyt and Tsunami malware in all models, while having lower misclassification rates for Mirai malware. Further, the RF-model is considered the least robust model, as shown in Figure 10a, as malicious samples were classified as benign.

Even though RF-based models provide the best classification performance on the clean dataset, this does not hold true under adversarial settings. Our evaluation shows that using a CNN-based model is noticeably better, considering a loss of $< 2\%$ performance on clean samples for both malware detection and classification tasks, while delivering higher robustness against GEA and SGEA.

6 DL-FHMC: Coping with AEs

Machine learning methods for malware detection and classification, *e.g.,* DL-SSMC, are susceptible to AEs and fall short of delivering a robust system against adversarial settings, as shown in §5. This motivates us to explore methods and alternative designs to cope with such vulnerabilities to adversarial attacks. In this section, we propose DL-FHMC, **F**ine-grained **H**ierarchical Learning for **M**alware **C**lassification, a robust system for malware detection and classification that leverages deep learning on a fine-grained and hierarchical manner to detect malicious behaviors.

6.1 DL-FHMC: System Design

The design of DL-FHMC consists of five components as illustrated in Figure 11. Description of each component is in the following.

- **CFG Extraction.** This component is responsible for extracting the CFGs of the software samples using Radare2, and presenting them as labeled CFGs for further analysis.
- **Feature Extraction.** The feature extraction component calculates 23 algorithmic features from the samples' CFGs. Details of the extracted features are in §4.1 and Table 2.
- **Malware Detection.** The malware detection component utilizes the IoT malware detection models as in Table 3. The purpose of this model is to classify the samples into malware and benign. Samples classified as benign are directed to the suspicious behavior detection process, while samples classified as malware are directed to the classification model.
- **Malware Classification.** This component is fed by the samples classified as malware by the malware detection component. The goal of this component is to classify the sample into three IoT malicious families, *i.e.,* Gafgyt, Mirai, and Tsunami. The design and architecture of this component is similar to the ones used for the classification task in DL-SSMC, except that it only contain the malicious classes (*i.e.,* no benign label).
- **Suspicious Behavior Detector.** This component detects a potential suspicious behavior within the samples. Since the adversary may generate AEs with the purpose of fooling the system in assigning them to benign class, this component further investigates the potential of suspicious behavior within the benign-classified sample using the extracted CFG.

6.2 Suspicious Behavior Detector

Suspicious Behavior Detector is a graph mining-based technique to investigate suspicious malicious patterns within software samples classified as benign. Figure 12 highlights the design of the Suspicious Behavior Detector, which consists of four modules, ❶ subgraphs mining, ❷ pattern selection, ❸ data representation, and ❹ suspicious behavior detection model. In the following, we describe each module.

❶ **Subgraphs Mining:** This module extracts common subgraphs within each IoT malware family. Using gSpan, we extracted and collected frequent subgraphs of a size range between 5 to 20 nodes from each malicious family. In particular, we used the gSpan algorithm to extract subgraphs from the training samples of each malicious family. This process took more than 160 hours to finish and resulted in over 2,150,170 patterns distributed as: 22,953 for Gafgyt, 127,217 for Mirai, and over 2,000,000 for

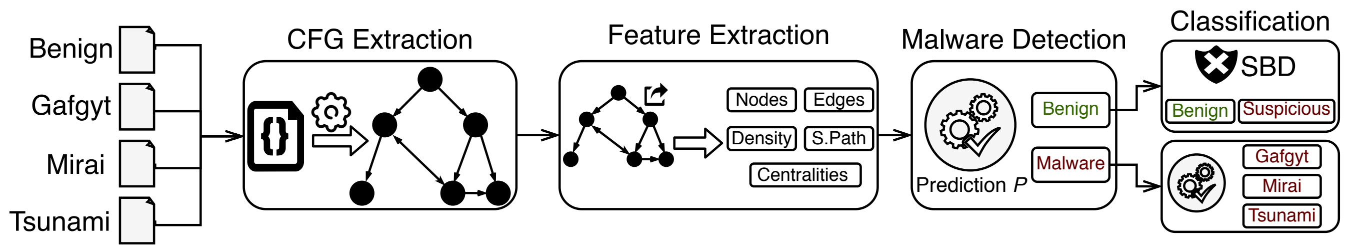


Fig. 11: DL-FHMC system flow. First, corresponding CFGs of the IoT software are extracted, then, 23 algorithmic features are extracted from the CFGs. Afterward, an IoT malware detection system classifies samples into benign and malware, all malware samples are directed to IoT malware classification system, while benign samples are directed into suspicious behavior detection system (SBD) for further investigation.

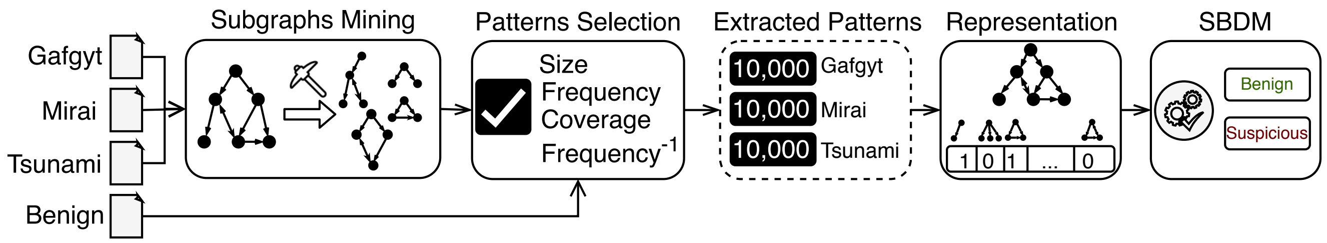


Fig. 12: Suspicious behavior detection system design. The design consists of four modules, a subgraphs mining module to extract frequent subgraphs from three IoT malicious families. Afterward, the subgraphs are ranked by the pattern selection module, where the top 10,000 patterns of each malicious family are selected. Further, the CFG of each sample is redirected to the Suspicious Behavior Detector, and represented as a vector of size $30,000$. The vector representation is fed to Suspicious Behavior Detector Model (SBDM) to be classified into benign and suspicious.

Tsunami families. The extracted frequent subgraphs (patterns) for each malicious family are then subjected to further analysis.

❷ **Pattern Selection:** This module ranks the extracted patterns based on four factors: pattern size, frequency, coverage, and inverse frequency. For example, large patterns are assigned higher value since they are distinctive and more likely to be unique to their family. Moreover, large patterns can be further decomposed into smaller patterns. Further, the number of occurrences of a pattern within a malicious family is considered as an indicator of its maliciousness. On the other hand, less frequent patterns are more likely to be function-oriented and solely contribute to the functionality of the code rather than the general behavior of the malware family. Therefore, we excluded all patterns that occurred in less than 5% of the targeted family samples. The coverage of the pattern is defined as $\sum_{i=1}^{n} 1/\text{occurrence}_i - 1$, where n is a set of samples in which the pattern occurred, and $occurrence_i$ is the number of patterns contained within the sample i. For example, if a sample contains only one pattern, that pattern will have the highest rank. In addition, we compute the number of occurrences for each pattern in the benign training samples. Note that benign samples may have patterns similar to the ones in the malware due to the abstract nature of the CFG and the considered size and functionality of patterns. To ensure that all patterns hold some behavioral characteristics of the malicious family, we excluded all malicious patterns that appeared in more than ten benign samples. We filtered the patterns and selected the top 10,000 ranked patterns from each family to be its representative patterns. This results in a total of 30,000 malware patterns for the three malware families. We denote this set of patterns as P.

❸ **Data Representation:** To investigate an IoT software, we find whether each of the selected 30,000 patterns is a subgraph in the CFG of the software using the VF2 subgraph isomorphism algorithm [35]. Each sample is represented as a binary vector in the space of the patterns extracted in the previous module, i.e., $v \in \{0,1\}^{|P|}$. Specifically, we represent each sample by a hot-encoding vector v of size 30,000, where $v_i = 1$ if the i^{th} pattern is a subgraph of the sample's CFG, i.e., $v_i = 1$ if $p_i \subseteq G$, $p_i \in P$. Time-wise, representing a software's CFG as a hot-encoding vector may require several minutes and up to several hours, depending on its size (number of nodes) and structure.

❹ **Suspicious Behavior Detection Model:** Suspicious Behavior Detector model is a machine learning model trained on the feature representations extracted from the training dataset shown in Table 1. The goal of this module is to investigate suspicious behavior within the sample. If the sample is classified as suspicious, further analysis is required by an analyst or dynamic analysis approach.

Experimental Setup. We trained the Suspicious Behavior Detector model on the feature representation of the training dataset, where all malicious samples are labeled as suspicious. We did not incorporate AEs in the training process, as doing so may bias the evaluation of the system toward samples generated using the same approach (*e.g.,* SGEA). Further, we generated AEs using both GEA and SGEA, to force the detection model to misclassify the IoT malware samples as benign, thereby, directing the samples to the suspicious behavior detector component. The generation of the AEs is similar to the process discussed in §5.

6.3 DL-FHMC: Evaluation and Results

The DL-FHMC system aims to establish a robust malware classification approach through hierarchical levels of abstractions. In the following, we evaluation the baseline classification performance, alongside the robustness of DL-FHMC.

DL-FHMC: Baseline Performance. The first task of the system is malware detection and classification. For the malware detection module, we used the same approach as in section 5, achieving the same baseline results (Table 3). All software classified as malware are then forwarded to the classification module, where they are labeled as Gafgyt, Mirai, or Tsunami. Table 9 shows the overall performance of DL-FHMC on the clean dataset (*i.e.,* non-adversarial). We report the overall accuracy and F-1 score, along with the individual classes true positive rates. We note that as we

TABLE 9: DL-FHMC classifier evaluation (%) on clean dataset for IoT malware classification task.

Architecture	Acc.	F-1	Benign	Gafgyt	Mirai	Tsunami
RF	97.67	97.66	99.16	97.66	98.00	92.69
DNN	95.74	95.73	95.33	97.33	95.33	93.60
CNN	96.38	96.38	97.00	97.16	95.66	94.52

TABLE 10: DL-FHMC Suspicious Behavior Detector evaluation (%) on benign and adversarial samples. DO refers to Data origin.

DO	FPR	GEA			SGEA	Overall
		Small	Median	Large		
RF	1	74.84	70.04	53.06	74.26	74.24
	3	84.86	79.10	100	79.31	88.05
	5	86.79	82.48	100	81.90	89.23
	10	89.24	97.40	100	86.90	92.71
DNN	1	63.28	59.50	49.76	64.90	67.28
	3	63.81	59.66	49.28	65.72	67.09
	5	63.72	59.40	48.79	65.78	66.54
	10	63.21	58.56	47.52	65.54	64.96
CNN	1	62.70	55.50	49.76	56.77	64.74
	3	62.87	55.40	49.28	57.30	64.37
	5	62.69	55.04	48.79	57.16	63.73
	10	62.06	54.02	47.53	56.49	62.02

use the malware detector as of DL-SSMC, the model is susceptible to adversarial attacks (*i.e.,* GEA and SGEA). All benign samples, alongside all successful AEs, are forwarded to the suspicious behavior detection module.

DL-FHMC: Suspicious Behavior Detection Task. This component aims to further investigate the benign-classified samples based on patterns extracted from their structural components. The task of suspicious behavior detector is to determine whether a given sample is signaling a suspicion of malicious behavior, and therefore it is operating as an AEs detection technique. We evaluate the Suspicious Behavior Detector using the original benign samples and malicious AEs. As shown in Table 10, RF-based detector achieves an overall performance of 89.23% with a benign accuracy of 95% (*i.e.,* false positive rate of 5%), while achieving a performance of 92.71% with a false positive rate of 10%. We note that the DNN- and CNN-based detectors are not effective as modalities for suspicious behavior detection.

While GEA large graph-based AEs achieve 100% misclassification rate, DL-FHMC can detect them with 100% accuracy. Further, while the detection performance for small GEA embeddings is relatively lower than the other configurations, the smaller the x_{sel} graph size is, the lower the success rate of the attack. Figure 13 shows the performance of the detector with different false positive rates ($1 -$ benign detection accuracy).

These results show that using DL-FHMC enables systematic methods of coping with adversarial manipulation to malware. When suspicious behavior is detected for a given sample, other methods can be adopted to further analyze the sample in order to provide a secure and robust evaluation of malicious activities. In general, using CNN-based architecture for baseline malware detection and classification tasks, while using RF-based model for suspicious behavior detection provides the best trade-off between the accuracy and robustness, as it minimizes the number of samples misclassified by the malware detector, and forward fewer samples toward the suspicious behavior detection system.

7 Discussion

DL-FHMC: Cost of Security. The security of machine learning algorithms is important for adoption in many applications. In the malware classification field, AEs pose critical security implications as emerging studies have shown that AEs can fool the machine learning-based malware detection system [1], [20], [36]. However, limited studies have investigated potential defenses. In this work, we show that launching adversarial attacks against malware detection systems can lead to a misclassification rate of as high as 100%. To cope with such adversarial settings and capabilities, we introduced DL-FHMC that operates on multiple levels of behavioral analysis of the software to ensure its security. Since AEs are derived from a combination of benign and malware components, detecting them is a challenging task and often comes at the cost of misclassifying a portion of benign samples as malicious, hence producing false alarms. For example, in our experiments, we show the trade-off (*i.e.,* the cost) of performance of the suspicious behavior detector and the benign samples misclassified as adversarial, know as the sensitivity of the detector, illustrated in Figure 13.

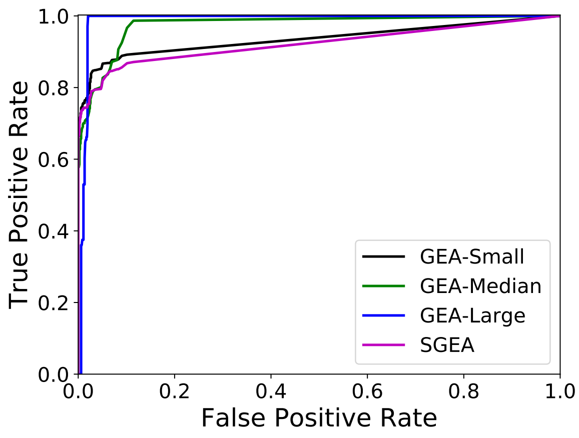


Fig. 13: DL-FHMC: RF-based suspicious behavior detector ROC performance over GEA and SGEA attacks.

DL-FHMC Robustness. The suspicious behavior detector in DL-FHMC consists of a machine/deep learning model trained on the vector representations obtained by checking the existence of predefined list of malicious subgraphs extracted and filtered using a large dataset of IoT malware. Using the VF2 subgraph isomorphism algorithm [35] on a list of 30,000 malicious subgraphs, the vector representation of the CFG is generated indicating whether malicious patterns exist in the CFG. Ideally, the suspicious behavior detector represents the benign samples into a vector of all zeros. In our dataset, $\approx 85\%$ of the benign samples are represented as a vector of all zeros, indicating that none of the malicious patterns were captured within them, *i.e.,* none of 30,000 malicious subgraph structure were found within their CFGs.

Considering the highly-accurate current state-of-the-art methods for detecting malware samples, our approach can be viewed as a robust layer in top of such methods. We argue that in the context of adversarial attacks, malware detection systems should be robust against perturbation to the original samples. Otherwise, they will suffer from the effects of AEs (i.e., misclassification). Since running the VF2 subgraph isomorphism algorithm against the 30,000 subgraphs introduces a computational overhead, a malware detection system benefits from our approach by testing samples that are classified as benign (with low classification confidences) indicating the possibility of adversarial scenario for evading the malware detection process.

Since the suspicious behavior detector in DL-FHMC uses a comprehensive list of malicious patterns, generating successful AEs against DL-FHMC requires modifying the functionality of the malware to conceal malicious patterns (*i.e.,* subgraphs) that exist in the CFG. This requires applying direct modifications to the control flow of the program, which in turn, contrasts the practicality and functionality requirements of a practical AE.

We note that detecting suspicious AEs is not a trivial process even with using a subgraph isomorphism matching algorithm, and it requires careful considerations in designing the system. For example, due to the abstract graph modality, 15% of the benign samples have suspicious subgraph structures within their CFGs. This does not, in most cases, indicate an embedded malicious functionality, but due to the diversity of the benign dataset, control flow structures similar to the ones occurring in the malicious samples may be found. This motivates using machine/deep learning methods to determine whether a sample is malicious considering the generated binary vector representation of the 30,000 subgraphs. Moreover, considering the evolution of malware and the emergence of new variants, the list of malicious subgraphs should be continuously updated to incorporates emerging patterns.

Suspicious Behavior Detector as an Individual Modality. The suspicious behavior detector is a machine learning-based CFG subgraph malicious patterns detector. It has its own feature space and representation, and independent model. While in this study we consider the detection and classification modules similar to the ones considered in DL-SSMC, the suspicious behavior detector is not exclusive to them only. In reality, following the same process, the detection and classification modules can be replaced with models that operate under different data representations. Then, forwarding all samples classified as benign to the suspicious behavior detector enables the AEs detection.

CFG for Malware Classification. Using CFG-based representations for malware detection and classifications address different challenges that may be raised by other representation techniques [37]. For instance, binary representations are susceptible to binary padding and injection. However, the added binaries are not typically executed, and therefore will not appear in the extracted CFG. Similarly, modifications on the header of the file, and stripping the binaries will not affect the final CFG. In general, changes that produce decision branches, such as conditions and loops, are the only modifications that alter the extracted CFG. We argue that adding API and system calls will simulate the GEA and SGEA attacks, as they only result in introducing new nodes, while the original structure is maintained, and thus can be accurately detected by DL-FHMC. Generating CFG-based AEs assumes a more powerful adversary, and even though we do not address other attacks that generate AEs using the alteration of the malware binaries or code, our work extend the robustness against such methods as changing such methods do not conceal existing malicious patterns from the CFG.

Potential Investigation Technique: Dynamic Analysis. For accurate and fail-proof malware detection, every sample should be analyzed dynamically and filtered based on its behavior. However, dynamic analysis has its own downsides: ❶ It requires setting up of a sandboxed environment such that the execution of the malware does not impact the underlining host system. ❷ It is costly in terms of time and memory. These make the dynamic analysis techniques difficult to scale. Our proposed technique puts forward a static analysis-based fine-grained hierarchical approach towards malware detection. The samples classified as benign in the first phase are sent to the suspicious behavior detector for investigation using the deeper CFG based features (subgraphs). The samples that are detected as suspicious in the second phase may be forwarded to dynamic analysis for further investigation. Statistically, and assuming a false positive rate of 5%, only 2.52% (51 out of 2,019) of the normal samples, *i.e.,* non-adversarial, were identified as the ones to be directed for dynamic analysis, thereby reducing the load on the dynamic analysis technique, hence overcoming its potential difficulties.

Binary Obfuscation. Malware authors often use different packing techniques, *e.g.,* Ultimate Packer for Executables (UPX), to obfuscate different parts of the malware code base, such as functions and strings. In obfuscated functions, the CFG would differ from the actual unpacked malware. Thereby, the detector should be aware of obfuscation and can accurately classify the obfuscated software and examining the behavior of packed and unpacked software.

8 Related Work

Malware Analysis. While efforts have been put towards malware analysis and detection in general, IoT malware robustness analysis still lacks exploring and investigation. Among the IoT malware studies, efforts towards the analysis and detection of malicious software are limited, particularly, from the lens of CFG. ManXu *et al.* [38] proposed a CNN-based malware detection system for the Android application from the semantic representation of the graph (*i.e.,* control and data flow graphs representations). In addition, Yang *et al.* [39] identified and detected Android malicious behaviors throughout generating two-level behavioral representations built from the CFG graph and call graphs of the program. Allix *et al.* [40] designed multiple machine learning classifiers to detect Android malware using different textual representations extracted from the applications' CFGs. Further, Alasmary *et al.* [33] conducted an in-depth CFG-based comparative study for the Android and IoT malware. Similarly, Pa *et al.* [41] established the first IoT honeypot and sandbox system, called IoTPOT, that run over eight CPU architectures to capture the IoT attacks running over Telnet protocol. Similarly, Caselden *et al.* [42] built an algorithm that generates an attack from the representation of the hybrid information and CFG applied to the program binaries. Alam *et al.* [43] proposed a metamorphic malware analysis and detection system that uses two different techniques that match the CFGs of small malware and then address the change in the opcodes frequencies. Moreover, Tamersoy *et al.* [44] proposed a malware detection algorithm that identifies the executable files of the malware and then computes the similarities between them to partial dataset files from the Norton Community Watch. Then, they construct graphs based on the measurement of inter-relationship between these files. In addition, Wuchner *et al.* [45] proposed a graph-based detection system that uses quantitative data flow graphs generated from the system calls, and uses the graph node properties, i.e., centrality metric, as a feature vector for the classification between malicious and benign programs. Moreover, they extended the work by using a compression-based mining technique applied to the quantitative data flow graphs for malware detection [46]. Moreover, Cen *et al.* [47] used Android API calls as features extracted from the decompiled source code of the software, and proposed a probabilistic logistic regression-based model for malware detection.

Furthermore, Qiu *et al.* [48] surveyed existing machine/deep learning-based android malware detection and classification systems. Their study shows a consistent trend of using neural network-based architectures for extracting deep representations and characteristics of the Android malware for the detection and classification tasks, which provide an improvement in comparison to the handcrafted features.

Adversarial Machine Learning. Machine/deep learning networks are widely used in security-related tasks, including malware detection [4], [11], [14], [49]. However, it has been shown that deep learning-based models are vulnerable against adversarial attacks [50]. Given that, it should be noted that such a behavior can be a critical issue in malware detection systems, where misclassifying malware as benign may result in disastrous consequences [20], [51]. Various adversarial machine learning attack methods in the context of image classification have been introduced to generate AEs. For example, Goodfellow *et al.* [16] introduced FGSM, a family of fast method attacks to generate AEs that forces the model to misclassification. In addition, Carlini *et al.* [52] proposed three L-norm-based adversarial attacks, known as C&W adversarial attacks, to investigate the robustness of neural networks and existing adversarial defenses. Similarly, Moosavi *et al.* [17] proposed DeepFool, an L_2 distance-based adversarial iterative method to generate AEs with minimal perturbation. Further, a critical application of the AEs is malware detection. Recent studies investigated generating AEs in the context of malware detection [36]. For instance, Grosse *et al.* [19] implemented an augmented adversarial crafting algorithm to generate AEs, misleading a CNN-based classifier to misclassify 63% of the malware samples to benign. Additionally, in the context of Android malware detection, Chen *et al.* [53] proposed a novel approach to evade the Android malware detection systems by applying optimal perturbations onto Android APK using a substitute model, utilizing the transferability characteristics of the AEs. This allows generating AEs to non-differentiable models, such as support vector machines and random forest. Applying perturbation directly onto APK's Dalvik bytecode, they achieved a performance degradation of more than 95% against two state-of-the-art detection approaches, MaMaDroid [54] and Drebin [55].

The detection of the AEs is challenging [56]. While work on detecting AEs in the context of IoT malware detection is very limit, multiple studies attempt to detect them in the context of image classification [57], [58], [59], achieving detection accuracy of 20% to 90%. In this study, we implemented DL-FHMC, a graph mining-based fine-grained hierarchical learning approach for suspicious behavior detection, achieving an overall performance of up to 92.71% in detecting CFG-based AEs.

9 Conclusion

This work introduces DL-FHMC, a novel hierarchical approach for robust malware detection and classification with AEs detection. To set out, first, an in-depth analysis of malware binaries is conducted through constructing abstract structures using CFG, which are analyzed from multiple aspects, such as the number of nodes and edges, as well as graph algorithmic constructs, such as average shortest path, betweenness, closeness, density, etc. Then, we evaluate the robustness of the traditional CFG-based IoT malware classification approaches against GEA and SGEA, achieving a misclassification rate of up to 100%. To address this, we use different graph mining techniques, CORK and gSpan, to extract malicious discriminative graphs from the malicious software, and use it as a modality to detect malicious behavior. Through our evaluation, DL-FHMC achieves a high malware detection and classification accuracy, as well as AEs detection performance under different GEA and SGEA configurations, with an overall AEs detection performance of up to 92.71%.

References

[1] A. Abusnaina, A. Khormali, H. Alasmary, J. Park, A. Anwar, and A. Mohaisen, "Adversarial learning attacks on graph-based iot malware detection systems," in *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*, 2019.

[2] Ericsson. (2018) Erisson mobility report. Available at [Online]: http://ericsson.com/en/press-releases/2018/6/5g-on-a-roll-cellular-iot-deployments-ramping-up--ericsson-mobility-report.

[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *Proceedings of the 26th USENIX Security Symposium*, 2017, pp. 1093–1110.

[4] A. Mohaisen, O. Alrawi, and M. Mohaisen, "AMAL: high-fidelity, behavior-based automated malware analysis and classification," *Computers & Security*, vol. 52, pp. 251–266, 2015.

[5] A. Gerber. (Retrieved, 2017) Connecting all the things in the Internet of Things. [Online]. Available: https://ibm.co/2qMx97a

[6] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "ContexloT: Towards providing contextual integrity to appified IoT platforms," in *Proceedings of the 24th Annual Network and Distributed System Security Symposium, NDSS*, 2017, pp. 1–15.

[7] A. Azmoodeh, A. Dehghantanha, and K.-K. R. Choo, "Robust malware detection for Internet Of (Battlefield) Things devices using deep eigenspace learning," *IEEE Transactions on Sustainable Computing*, vol. 4, no. 1, pp. 88–95, 2019.

[8] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! A case study on android malware detection," *IEEE Transaction on Dependable and Secure Computing*, vol. 16, no. 4, pp. 711–724, 2019.

[9] S. Siby, R. R. Maiti, and N. O. Tippenhauer, "IoTScanner: Detecting privacy threats in IoT neighborhoods," in *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security*, 2017, pp. 23–30.

[10] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: effective and efficient behavior-based android malware detection and prevention," *IEEE Transaction on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2018.

[11] H. Alasmary, A. Khormali, A. Anwar, J. Park, J. Choi, A. Abusnaina, A. Awad, D. Nyang, and A. Mohaisen, "Analyzing and detecting emerging internet of things malware: a graph-based approach," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8977–8988, 2019.

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

[13] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 116–127.

[14] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of DGA-based malware," in *Proceedings of the 21th USENIX Security Symposium*, 2012, pp. 491–506.

[15] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the ACM on Asia Conference on Computer and Communications Security, AsiaCCS*, 2017, pp. 506–519.

[16] C. S. Ian J. Goodfellow, Jonathon Shlens, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations.*, 2015, pp. 1–11.

[17] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A simple and accurate method to fool deep neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2574–2582.

[18] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016, pp. 372–387.

[19] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. D. McDaniel, "Adversarial examples for malware detection," in *Computer Security - ESORICS - 22nd European Symposium on Research in Computer Security*, 2017, pp. 62–79.

[20] A. Abusnaina, H. Alasmary, M. Abuhamad, S. Salem, D. Nyang, and A. Mohaisen, "Subgraph-based adversarial examples against graph-based iot malware detection systems," in *International Conference on Computational Data and Social Networks*. Springer, 2019, pp. 268–281.

[21] Q. Zhang and D. S. Reeves, "Metaaware: Identifying metamorphic malware," in *Proceedings of the Twenty-Third Annual Computer Security Applications Conference, ACSAC*, 2007, pp. 411–420.

[22] Developers. (Retrieved, 2018) the ultimate packer for executables. [Online]. Available: https://upx.github.io/

[23] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *IEEE Security & Privacy*, vol. 5, no. 2, pp. 32–39, 2007.

[24] B. Wang, Y. Yao, B. Viswanath, H. Zheng, and B. Y. Zhao, "With great training comes great vulnerability: Practical attacks against transfer learning," in *Proceedings of the 27th USENIX Security Symposium, USENIX Security 2018*, 2018, pp. 1281–1297.

[25] Developers. (Retrieved, 2019) Radare2. [Online]. Available: http://www.radare.org/r/

[26] M. Thoma, H. Cheng, A. Gretton, J. Han, H. Kriegel, A. J. Smola, L. Song, P. S. Yu, X. Yan, and K. M. Borgwardt, "Discriminative frequent subgraph mining with optimality guarantees," *Statistical Analysis and Data Mining*, vol. 3, no. 5, pp. 302–318, 2010.

[27] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Proceedings of the 2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, 2002, pp. 721–724.

[28] Developers. (2019) Cyberiocs. Available at [Online]: https://freeiocs.cyberiocs.pro/.

[29] ——. (2019) VirusTotal. Available at [Online]: https://www.virustotal.com.

[30] VirusShare. [Online]. Available: https://virusshare.com/

[31] Developers. (2019) Github. Available at [Online]: https://github.com/.

[32] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "AVclass: A tool for massive malware labeling," in *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses, RAID*, 2016, pp. 230–253.

[33] H. Alasmary, A. Anwar, J. Park, J. Choi, D. Nyang, and A. Mohaisen, "Graph-based comparison of IoT and android malware," in *Proceedings of the 7th International Conference on Computational Data and Social Networks, CSoNet*, 2018, pp. 259–272.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[35] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (Sub)graph isomorphism algorithm for matching large graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

[36] O. Suciu, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," in *2019 IEEE Security and Privacy Workshops, SP Workshops*, 2019, pp. 8–14.

[37] D. Park and B. Yener, "A survey on practical adversarial examples for malware classifiers," *arXiv preprint arXiv:2011.05973*, 2020.

[38] Z. Xu, K. Ren, S. Qin, and F. Craciun, "CDGDroid: Android malware detection based on deep learning using CFG and DFG," in *Proceedings of the 20th International Conference on Formal Engineering Methods, ICFEM*, 2018, pp. 177–193.

[39] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. A. Porras, "DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in android applications," in *Proceedings of the 19th European Symposium on Research in Computer Security*, 2014, pp. 163–182.

[40] K. Allix, T. F. Bissyandé, Q. Jérome, J. Klein, R. State, and Y. L. Traon, "Empirical assessment of machine learning-based malware detectors for android - measuring the gap between in-the-lab and in-the-wild validation scenarios," *Empirical Software Engineering*, vol. 21, no. 1, pp. 183–211, 2016.

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

[42] D. Caselden, A. Bazhanyuk, M. Payer, S. McCamant, and D. Song, "HI-CFG: construction by binary analysis and application to attack polymorphism," in *Proceedings of the 18th European Symposium on Research in Computer Security*. Springer, 2013, pp. 164–181.

[43] S. Alam, R. N. Horspool, I. Traoré, and I. Sogukpinar, "A framework for metamorphic malware analysis and real-time detection," *Computers & Security*, vol. 48, pp. 212–233, 2015.

[44] A. Tamersoy, K. A. Roundy, and D. H. Chau, "Guilt by association: large scale malware detection by mining file-relation graphs," in *Proceedings of the the 20th ACM International Conference on Knowledge Discovery and Data Mining, KDD*, 2014, pp. 1524–1533.

[45] T. Wüchner, M. Ochoa, and A. Pretschner, "Robust and effective malware detection through quantitative data flow graph metrics," in *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment Conference, DIMVA*, 2015, pp. 98–118.

[46] T. Wüchner, A. Cislak, M. Ochoa, and A. Pretschner, "Leveraging compression-based graph mining for behavior-based malware detection," *IEEE Transaction on Dependable and Secure Computing*, vol. 16, no. 1, pp. 99–112, 2019.

[47] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for android malware detection with decompiled source code," *IEEE Transaction on Dependable and Secure Computing*, vol. 12, no. 4, pp. 400–412, 2015.

[48] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, "A survey of android malware detection with deep neural models," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–36, 2020.

[49] A. Mohaisen and O. Alrawi, "Unveiling zeus: automated classification of malware samples," in *Proceedings of the 22nd International World Wide Web Conference, WWW*, 2013, pp. 829–832.

[50] T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae, and S. Ishii, "Distributional smoothing with virtual adversarial training," in *International Conference on Learning Representations.*, 2016, pp. 1–12.

[51] A. Abusnaina, D. Nyang, M. Yuksel, and A. Mohaisen, "Examining the security of ddos detection systems in software defined networks," in *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies*, 2019, pp. 49–50.

[52] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2017, pp. 39–57.

[53] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android hiv: A study of repackaging malware for evading machine-learning detection," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 987–1001, 2019.

[54] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version)," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 2, pp. 1–34, 2019.

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

[56] N. Carlini and D. A. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS*, 2017, pp. 3–14.

[57] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," in *25th Annual Network and Distributed System Security Symposium, NDSS*, 2018.

[58] X. Li and F. Li, "Adversarial examples detection in deep networks with convolutional filter statistics," in *IEEE International Conference on Computer Vision, ICCV*, 2017, pp. 5775–5783.

[59] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," in *5th International Conference on Learning Representations, ICLR*, 2017.

Ahmed Abusnaina is a Ph.D. student in the Department of Computer Science at the University of Central Florida. He obtained his B.Sc. in Computer Engineering from An-Najah National University, Palestine, in 2018. His research interests include software security, machine learning, and adversarial machine learning.

Mohammed Abuhamad received a Ph.D. degree in Computer Science from the University of Central Florida in 2020. He also received a Ph.D. degree in Electrical and Computer Engineering from INHA University in 2020. He is currently an assistant professor of Computer Science at Loyola University Chicago. His research interests include AI/Deep-Learning-based Applications in Information Security, Software and Mobile/IoT Security, and Adversarial Machine Learning.

Afsah Anwar is a Ph.D. candidate in the Department of Computer Science at the University of Central Florida. He obtained his B.S. from Jamia Millia Islamia University, New Delhi, India, in 2014. Before starting his Ph.D., Afsah was working as a Data Analyst (C) for Apple. His research interests include binary analysis, vulnerability analysis, and malware analysis.

Hisham Alasmary is an Assistant Professor at King Khalid University. He obtained his Ph.D. from the Department of Computer Science at the University of Central Florida in 2020, and his M.Sc. degree in Computer Science from The George Washington University, in Washington, D.C., USA, in 2016. His research interests include Software Security, IoT Security and Privacy, ML/DL Applications in Information Security, and Adversarial Machine Learning.

Rhongho Jang received his Ph.D. in the department of computer science at the University of Central Florida, in 2020. He also received his B.S., M.E, and Ph.D. (first) from the Inha University of South Korea in 2013, 2015, and 2020, respectively. He is currently an assistant professor in the department of computer science at Wayne State University. His research interests lie in the area of software defined networks, network security, traffic measurement, and mobile security in general.

Saeed Salem received his Ph.D. in computer science from Rensselaer Polytechnic Institute, New York. He is currently an associate professor at North Dakota State University. Dr. Salem's research is in the broad areas of graph mining and machine learning with a focus on developing algorithms for mining frequent and significant graphs. Dr. Salem's group developed enumeration algorithms for mining all frequent subgraphs, cross-graph dense graphs, and approximate frequent subgraphs from heterogeneous graphs.

DaeHun Nyang received a B.Eng. degree in electronic engineering from Korea Advanced Institute of Science and Technology, M.S. and Ph.D. degrees in computer science from Yonsei University, Korea in 1994, 1996, and 2000 respectively. He has been a senior member of the engineering staff at Electronics and Telecommunications Research Institute, Korea, from 2000 to 2003. Since 2003, he has been a full professor at Computer Information Engineering Department of Inha University, Korea where he is also the founding director of the Information Security Research Laboratory. He is a member of the board of directors and an editorial board of ETRI Journal and also Korean Institute of Information Security and Cryptology. Dr. Nyang's research interests include AI-based security, network security, traffic measurement, privacy, usable security, biometrics and cryptography.

David Mohaisen earned his M.Sc. and Ph.D. degrees from the University of Minnesota in 2012. He is currently an Associate Professor at the University of Central Florida, where he directs the Security and Analytics Lab (SEAL). Before joining UCF in 2017, he was an Assistant Professor at SUNY Buffalo (2015–2017) and a Senior Research Scientist at Verisign Labs (2012–2015). His research interests are in the areas of networked systems security, online privacy, and measurements. He is an Associate Editor of IEEE TMC, IEEE TCC, and IEEE TPDS. He is a senior member of both ACM (2018) IEEE (2015), a Distinguished Speaker of ACM, and Distinguished Visitor of IEEE.