Subgraph-based Adversarial Examples Against Graph-based IoT Malware Detection Systems

Ahmed Abusnaina[1], Hisham Alasmary[1], Mohammed Abuhamad[1,2]
Saeed Salem[3], DaeHun Nyang[2], and Aziz Mohaisen[1]

[1]University of Central Florida, Orlando, FL 32816, USA
[2]Inha University, Incheon, South Korea
[3]North Dakota State University, Fargo, ND 58105
ahmed.abusnaina,hisham,abuhamad@Knights.ucf.edu
mohaisen@ucf.edu, nyang@inha.ac.kr, saeed.salem@ndsu.edu

Abstract. Internet of Things (IoT) has become widely adopted in many fields, including industry, social networks, health care, and smart homes, connecting billions of IoT devices through the internet. Understanding and studying IoT malware through analysis using various approaches, such as Control Flow Graph (CFG)-based features and then applying deep learning detection, are widely explored. In this study, we investigate the robustness of such models against adversarial attacks. Our approach crafts the adversarial IoT software using the Subgraph Embedding and Augmentation (SGEA) method that reduces the embedded size required to cause misclassification. Intensive experiments are conducted to evaluate the performance of the proposed method. We observed that SGEA approach is able to misclassify all IoT malware samples as benign by embedding an average size of 6.8 nodes. This highlights that the current detection systems are prone to adversarial examples attacks; thus, there is a need to build more robust systems to detect such manipulated features generated by adversarial examples.

Keywords: IoT Malware Detection, Adversarial Learning, Graph embedding

1 Introduction

Internet of Things (IoT) malware has emerged as one of the most challenging threats on the Internet today [1], and is expected to grow for many years to come. To cope with this threat, there has been a lot of works in the literature on the analysis, characterization and detection of IoT malware [2], falling under both static and dynamic analysis-based approaches [3]. One of the prominent static-based approaches to IoT analysis and detection uses abstract graph structures, such as the control flow graph (CFG) [4–6]. In using the CFGs for detecting IoT malware, defenders extract feature representations that are capable of identifying those malware, including various graph properties, such as the degree distribution, centrality measures, diameter, radius, etc. [4]. Those properties, represented as a feature vector, are used in tandem with machine learning algorithms to automate the labeling and detection of IoT malware samples.

As with other machine learning algorithms and applications, machine learning-based IoT malware detection algorithms are prone to manipulation. The rise of adversarial machine learning has highlight the fragile nature of those algorithms to perturbation

attacks that would lead to misclassification: an adversary can introduce a small modification to the input sample space that would make the classifier to identify a piece of malware as a benign sample (i.e., adversarial example; or AE). Indeed, there has been a large body of work exploring the generation of AEs in general image-based classification problems [7, 8] as well as in the context of malware classification [9–11].

In this work, we optimize the Graph Embedding and Augmentation (GEA), a recent work on generating AEs in the context of CFG-based malware detection [11]. GEA aims to inject a piece of code into a target sample to alter its graph representation and the resulting feature used by the machine learning algorithm. In this work, we introduce sub-GEA (SGEA), an AE generation algorithm that mines for discriminative patterns (subgraphs) from a targeted class, and embeds such subgraphs in a sample towards generating the AE. SGEA does not only result in a high misclassification rate, but achieves the essence of AE generation: a small perturbation (as measured by the perturbation size) to the sample to result in the misclassification.

Contributions. Our contributions in this paper are as follows. First, we propose SGEA, a graph embedding technique to generate AEs with reduced injection size. Second, we show the favorable performance of SGEA by comparing it to GEA for both IoT malware detection and classification. GEA and SGEA both generate adversarial IoT software through embedding representative target sample to the original CFG representation of the targeted sample, while maintaining its practicality and functionality. We evaluate the performance of the methods via intensive experiments showing the effectiveness of the approach in producing successful AEs.

Organization. In section 2, we discuss the related work. Then, the practical approach for generating practical adversarial IoT software is described in section 3. The performance of the proposed approach, evaluated through intensive experiments, are in section 4. Finally, we conclude our work in section 5.

2 Related Work

Static analysis using various methods, including and CFGs, is well explored. Wuchner *et al.* [12] proposed a graph-based classification system that uses features generated from the quantitative data flow graphs of system calls. Moreover, Alasmary *et al.* [4] analyzed Android and IoT malware based on CFG features and built a deep learning-based detection system for IoT malware utilizing these features. Caselden *et al.* [13] proposed an attack on the program binaries using static representations of hybrid information and CFG. Alam *et al.* [14] proposed a malware detection system that matches CFGs of small malware samples and addresses changes occurred in opcodes' frequencies using two methods. Bruschi *et al.* [15] proposed a CFG-based malware detection system to compare extracted CFGs to known malware samples CFGs and then detect the malware based on these graphs. Similarly, Yan *et al.* [16] classified malware samples using CFG-based representation with deep convolutional neural networks. Machine and deep learning algorithms are widely deployed in malware detection [3, 4, 17]. However, deep learning-based models are vulnerable to adversarial attacks [9]. As a result, current malware detection systems can be fooled to misclassify crafted malware samples that are generated by applying small perturbation to the malware resulting in disastrous consequences. For example, DeepFool attack was proposed by Moosavi *et al.* [8] that uses iterative methods of L_2 distance-based adversarial to generate AEs with minimal

perturbation. Also, Goodfellow *et al.* [7] proposed fast method attacks, FGSM, to generate AEs that fools the model. Moreover, three adversarial attacks, called C&W, were proposed by Carlini *et al.* [18] to explore the robustness of neural networks and existing defense techniques. Although AE generation approaches are well explored in image-based classifiers, limited research have been conducted on generating AE for malware samples [10, 19], such as GEA [11]

3 Generating Adversarial Examples

Adversarial examples are generated by applying perturbation to the input feature space, $x' = x + \epsilon$, where x is the input vector, and ϵ is the perturbation. The adversary aims to misclassify the output of the targeted model, altering $f(x) \neq f(x')$, where f is the model's output. To reduce the detectability of the generated AEs, ϵ is minimized while preserving the adversarial behavior, $f(x) \neq f(x + \delta)$, where $\delta = \epsilon_{min}$. To do so, multiple approaches have been proposed [8, 18]. In IoT malware, the process aims to generate realistic AEs that preserve the functionality of the original samples, although not maintained by the literature. Recently, Abusnaina *et al.* [11] proposed a new approach to highlight these issues, called Graph Embedding and Augmentation (GEA), in which the perturbation is applied at the code-level of the samples, ensuring the practicality of the generated AE. In this study, SGEA is an enhanced approach to generate AEs that reduces the perturbation overhead compared to GEA.

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

}
```

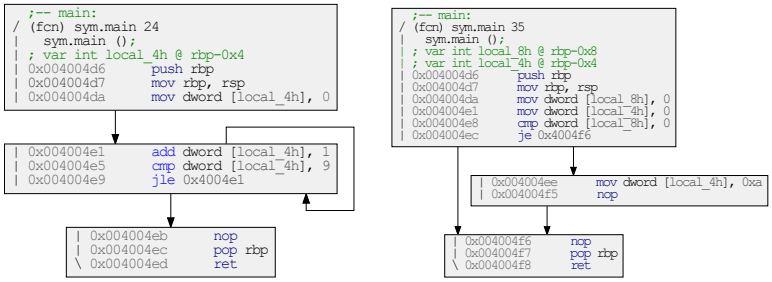
Listing 1.1: C script (original)

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

Listing 1.2: C script (original)

3.1 Graph Embedding and Augmentation (GEA)

GEA [11] generates practical AEs that maintain the functionality of the original IoT software while also achieving a high misclassification rate. This is done by combining the CFG of the original sample with a selected CFG (graph merging). **Practical Implementation.** GEA preserves the practicality of GEA by merging the original sample x_{org} with a selected target sample x_{sel}, which we highlight by an example. Listing 1.1 refers to the original sample script, while Listing 1.2 refers to the selected target sample script. The goal is to combine the two scripts while insuring that x_{sel} does not affect the functionality of x_{org}. Listing 1.3 shows the script after the combination. Note that the condition is set to execute only the functionality associated with x_{org} while preventing x_{sel} functionality from being executed. Figure 1a and Figure 1b show the corresponding graphs for x_{org} and x_{sel}, respectively. It can be seen that the combined graph in Figure 2 consists of the two aforementioned scripts sharing the same entry and exit nodes.



(a) Generated CFG (b) Targeted CFG

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

```c
#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 1.3: C script of combining original and selected samples

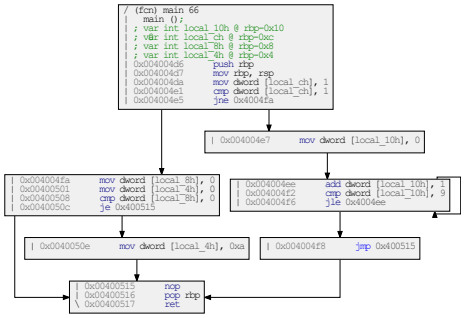


Fig. 2: The generated adversarial graph using GEA approach. Note that this graph is obtained logically by embedding the graph in Fig. 1b into the graph in Fig. 1a, although indirectly done by injecting the code listings as highlighted in Listings 1, 2, and 3.

3.2 Sub-GEA (SGEA)

While GEA combines original and selected targeted graphs of the samples to cause misclassification, this method reduces the injection size while preserving the same behavior, misclassification. To achieve that, we first convert the training samples of each class to CFGs using Radare2 [20], then, we extract discriminative subgraph patterns of each class using a correspondence-based quality criterion (CORK) [21]. This is done by extracting subgraphs that appear frequently in one class and less frequently in the other class. 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.
Let $D_S = \{G_i | S \subseteq G_i \ and \ G_i \in D\}$ denote the supporting graphs of S. More-

over, let D_S^+ and D_S^-, denote the supporting graphs of the subgraph in the positive graphs and negative graphs, respectively. The CORK algorithm 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 the graphs of one class and not once in the graphs of the other class; the quality score is 0. Pruning strategies, proposed based on the quality criterion in the CORK algorithm, are integrated in the gSpan algorithm to directly mine discriminative subgraphs. Once the set of discriminative subgraphs are mined using the CORK algorithm, we further employ the gSpan [22], a graph-based substructure pattern mining, for mining frequent subgraphs of size five nodes or higher.

Constructing an AE Then, we combine the original sample with the smallest extracted subgraph of the targeted class regarding nodes' number. If the generated CFG failed to be misclassified, another subgraph is selected in an ascending order with respect to the number of nodes in the collection of subgraphs. When a subgraph successfully achieves the misclassification, the operation ends. If no existing subgraph could cause misclassification, the original sample is returned and the operation fails. **Practical Implementation.** The process is done by combining the x_{org} with a selected discriminative subgraph x_{sel} extracted from the targeted class. Figure 3 shows the discriminative subgraph extracted from Gafgyt IoT malicious family. Listing 1.4 is an equivalent sample of C script to generate such subgraph, which can be combined with the original sample to generate the practical AE.

```
#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 1.4: C script of an example Gafgyt extracted subgraph

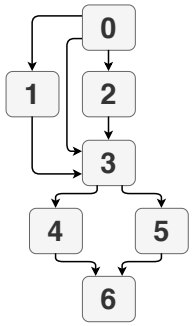


Fig. 3: Sample of extracted discriminative subgraph from Gafgyt malicious family.

4 Evaluation and Discussion

In this section, we evaluate the performance of the approaches over deep learning-based malware detection and classification systems trained over CFG-based features.

4.1 Dataset

To assess our proposed approaches, we started by gathering the dataset. To facilitate the evaluation, we gathered a dataset of binaries of two categories, IoT malicious and

Table 1: Distribution of IoT samples across the classes.

Class	# of Samples			% of Samples
	# Train	# Test	# Total	
Benign	2,400	600	3,000	34.60%
Gafgyt	2,400	600	3,000	34.60%
Mirai	1,927	481	2,408	27.68%
Tsunami	210	52	262	3.02%
Overall	6,937	1,733	8,670	100%

Table 2: Distribution of extracted features.

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

benign samples. Malicious samples are recent, in particular they were collected from the period of January 2018 to late February of 2019 from CyberIOCs [23]. Moreover, we assembled a dataset of benign samples from source files on GitHub [24].

Dataset Creation. Our dataset consists of 3,000 IoT benign samples and 5,670 IoT malware samples gathered from CyberIOCs [23]. We reverse-engineered the datasets using Radare2 [20], a reverse engineering framework that provides various analysis and automation capabilities, including disassembly. Upon disassembling the binary of each sample, benign or malicious, we extracted the corresponding CFG of each sample.

Ground Truth Class. To validate our benign and malicious samples, we uploaded them on *VirusTotal* [25] and gathered the scan results corresponding to each sample. Then, we used AVClass [26] to classify malicious samples to their corresponding families. We summarized our dataset in Table 1.

Moving forward, we find different algorithmic features of the CFGs corresponding to individual binaries. In particular, for each sample, we extract 23 various algorithmic features categorized into seven groups, as in [4]. Table 2 represents the feature category and the number of features in each category. The five features extracted from each of the four feature categories represent minimum, maximum, median, mean, and standard deviation values for the observed parameters.

4.2 Experimental Setup

IoT Malware Detection System The goal of our detection system is to recognize IoT malicious applications from benign. Therefore, we trained two deep learning models, Convolutional Neural Network (CNN)-based and Deep Neural Network (DNN)-based models, over the extracted CFG-based features. In this study, the input (X) of the model is a one dimensional (1D) vector of size 1×23 representing the extracted features.

CNN-based Design. We implemented the CNN architecture utilized in Abusnaina *et al.* [11]. Figure 4 shows the internal design of the CNN architecture and, although a standard design, more details can be found in the original paper [11].

DNN-based Design. The DNN-based model architecture consists of two consecutive fully connected dense layers of size 1×100 connected to the input vector, followed by a dropout with a probability of 0.25. Similarly, the output of the dropout function is fully connected with another two fully connected dense layers of size 1×100, followed by a dropout with a probability of 0.5. The output is then fed to the softmax layer to be evaluated based on the AR, FNR, FPR, to measure the performance of the model.

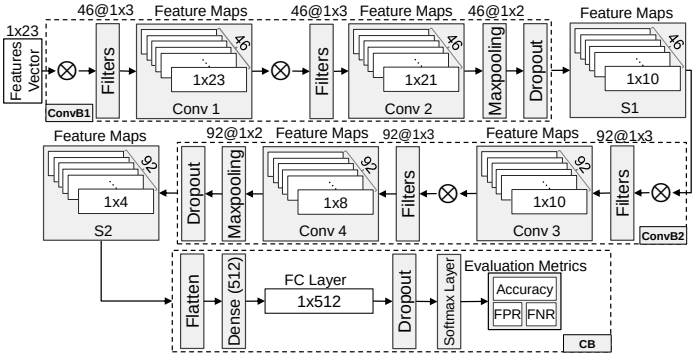


Fig. 4: The design of our CNN architecture.

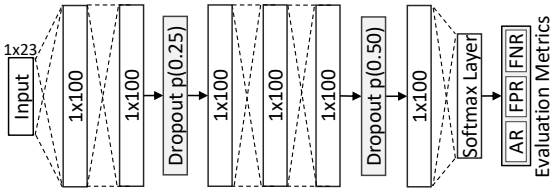


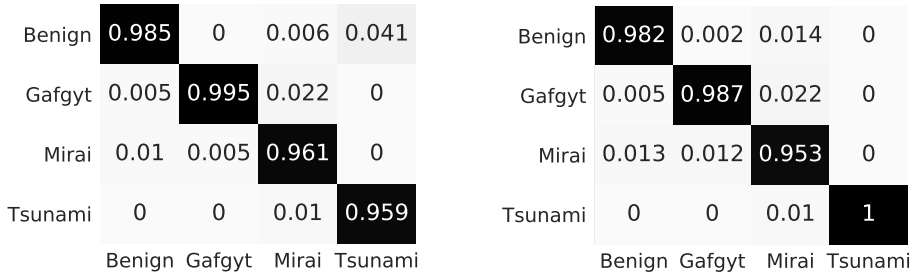
Fig. 5: The design of our DNN architecture.

We trained both models using 100 epochs with a batch size of 32. The architecture of the CNN and DNN designs are shown in Figure 4 and Figure 5, respectively.

IoT Malware Classification System. In addition to recognizing IoT malicious applications, the malware classification system distinguishes and classifies the malicious samples to their corresponding families. Similar to the detection system, we utilized the aforementioned CNN- and DNN-based architectures and trained two deep learning models to classify the samples.

GEA. Due to the nature of the extracted features, the applied changes on the CFG will be reflected in the features, regardless of the effects on the functionality and practicality of the original sample. Similar to [11], we selected six different sized graphs (minimum, median and maximum) from the benign and malware samples as x_{sel}, where the size is referred to as the number of nodes in the graph.

SGEA. While GEA modifies the CFG of the sample by simply connecting the selected graph with the original sample. SGEA connects a carefully generated subgraph with the original sample to cause misclassification, reducing the injected graph size. To generate the subgraph, we extracted the discriminative subgraph patterns from each class and their subgraphs of size five nodes or higher. Then, in order to reduce the injected graph size, we connected the original sample with the subgraph of the minimum size. If the generated AE misclassifies the classifier, the process succeeds, and the AE will be returned, if not, the next subgraph is selected, 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.



(a) CNN-based (b) DNN-based

Fig. 6: Confusion matrices of IoT malware classification systems.

4.3 Results and Discussion

Deep Learning-based IoT Malware Detection Systems. We designed two-class classifications, CNN- and DNN-based models, that distinguish IoT malware from IoT benign applications. The model is trained over 23 CFG-based features categorized into seven groups. More detailed information regarding the dataset is provided in subsection 4.1. We achieved a CNN- and DNN-based model accuracy rate of 98.96% and 98.67% with a False Negative Rate (FNR) of 0.88% and 1.41% and a False Positive Rate (FPR) of 1.33% and 1.16%, respectively.

Deep Learning-based IoT Malware Classification Systems. We designed four-class classifications, CNN- and DNN-based models, that are capable of classifying the malicious samples into their corresponding families. We achieved a CNN- and DNN-based model accuracy rate of 98.09% and 97.57%, respectively. Figure 6a and Figure 6b are the confusion matrices of the trained models.

GEA. We investigated the impact of the size of the graph on the misclassification rate. We selected three graphs, as targets, from each of the benign and malicious IoT software, and connected each of these target graphs with a graph of the other class to understand the impact of size on misclassification with GEA. The results for the IoT detection system are shown in Table 3. It can be observed that the misclassification rate increases when the number of nodes increases. In addition, the time needed to craft the AE is proportional to the size of the selected sample. In the IoT malware detection systems, we achieved a malware to benign misclassification rate of as high as 100% on both CNN- and DNN-based models, and a benign to malware misclassification rate of 60.22% and 60.89% on CNN- and DNN-based models, respectively. Table 4 shows the targeted and non-targeted misclassification rates over the IoT malware classification systems. Non-targeted misclassification indicates that after combining the original and selected samples, the original sample class changes. Moreover, if the new assigned label is the selected sample class, it is considered as targeted misclassification. Here, we achieved a targeted misclassification rate of 100% from all malicious families into benign on both CNN- and DNN-based models, highlighting the security issue in such systems.

SGEA. Similar to GEA, SGEA focuses on generating the adversarial desired output. In addition, SGEA reduces the size of injection by combining the original sample with a carefully selected subgraph. Table 5 shows the evaluation of SGEA against CNN- and DNN-based IoT malware detection systems. Notice that GEA outperforms SGEA

Table 3: GEA: Misclassification rate over IoT detection systems. MR refers to misclassification rate, whereas, CT refers to the crafting time in millisecond per sample.

Malware to benign MR.

Size	# Nodes	MR (%)		CT (ms)
		CNN	DNN	
Minimum	10	50.57	61.03	37.39
Median	23	99.64	98.76	40.46
Maximum	1075	**100**	**100**	6,430.66

Benign to malware MR.

Size	# Nodes	MR (%)		CT (ms)
		CNN	DNN	
Minimum	11	45.92	19.33	34.10
Median	43	**60.22**	59.90	56.83
Maximum	274	47.36	**60.89**	763.63

Table 4: GEA: Misclassification rate over IoT classification systems.

Class	# Nodes	Misclassification Rate				Crafting Time (ms)
		Non-targeted		Targeted		
		CNN	DNN	CNN	DNN	
Benign	10	48.72%	58.87%	45.89%	48.54%	37.39
	23	99.64%	99.55%	99.64%	99.38%	40.46
	1075	**100%**	**100%**	**100%**	**100%**	6,430.66
Gafgyt	13	24.62%	41.74%	0.17%	0.35%	32.36
	64	**66.81%**	**77.40%**	**15.97%**	**16.06%**	68.77
	155	54.63%	47.13%	8.03%	0.00%	125.15
Mirai	11	41.37%	37.22%	0.32%	0.80%	34.10
	48	62.30%	52.15%	12.46%	0.96%	56.84
	274	**95.60%**	**91.05%**	**93.45%**	**53.67%**	763.63
Tsunami	15	59.54%	60.73%	**0.12%**	**0.12%**	35.25
	59	63.95%	64.06%	0.00%	0.00%	59.93
	138	**66.74%**	**64.36%**	0.00%	0.00%	201.82

in benign to malware misclassification. However, SGEA achieves 100% malware to benign misclassification rate against CNN- and DNN-based models, with an average subgraph size of 6.8 nodes, outperforming the GEA approach.

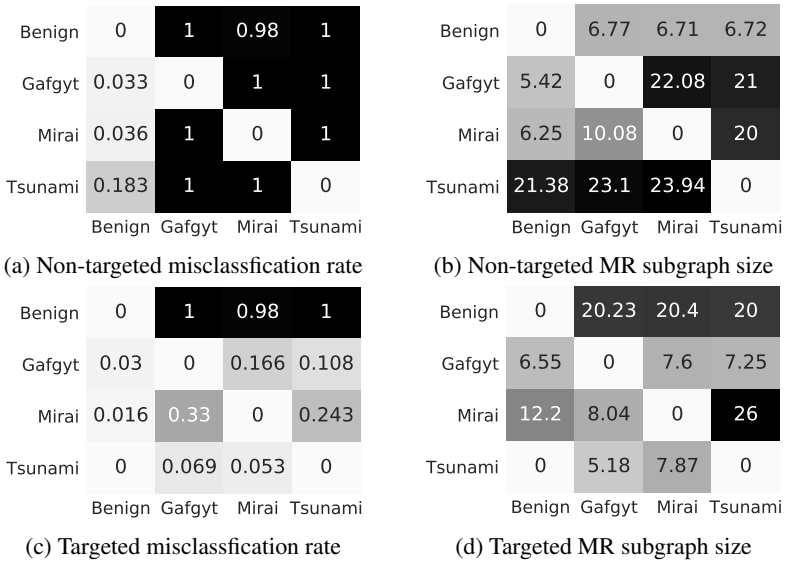
Figure 7 and Figure 8 show the evaluation of CNN- and DNN-based IoT malware classification systems against SGEA approach. Here, Figure 7a and Figure 7c represent the non-targeted and targeted misclassification rate over CNN-based model, respectively. Similarly, Figure 8a, and Figure 8c show the non-targeted and targeted misclassification rate over DNN-based model. Figure 7b, Figure 7d, Figure 8b and Figure 8d represent the average size of the connected subgraphs to generate the AEs over the CNN- and DNN-based models. For instance, SGEA approach successfully targeted misclassifies all Gafgyt test samples into benign over CNN-based classification model with an average subgraph size of 20.23 nodes, which is significantly better than GEA. Moreover, it misclassifies all Gafgyt test samples into other classes using discriminative subgraphs extracted from benign samples with an average size of 6.77 nodes. Notice that all classes have a high targeted misclassification rate towards the benign class and a low targeted misclassification rate from benign to malicious families and among malicious families. This behavior is caused by the nature of the benign samples, as they are diverse in characteristics and functionalities. However, malicious samples within the

Table 5: SGEA: IoT malware detection system evaluation. Here, MR refers to misclassification rate, AVG. Size refers to the overall average subgraph size used to achieve misclassification, and CT is the AEs crafting time per sample in seconds.

Benign to malware misclassification.			
Architecture	MR(%)	AVG. Size	CT (s)
CNN	22.22	10.15	2.57
DNN	33.88	11.09	2.23

Malware to benign misclassification.			
Architecture	MR(%)	AVG. Size	CT (s)
CNN	100	6.80	0.23
DNN	100	6.86	0.21

same family tend to have the same functionalities, resulting in high level of similarity in the extracted CFGs and their corresponding features.



(a) Non-targeted misclassfication rate

(b) Non-targeted MR subgraph size

(c) Targeted misclassfication rate

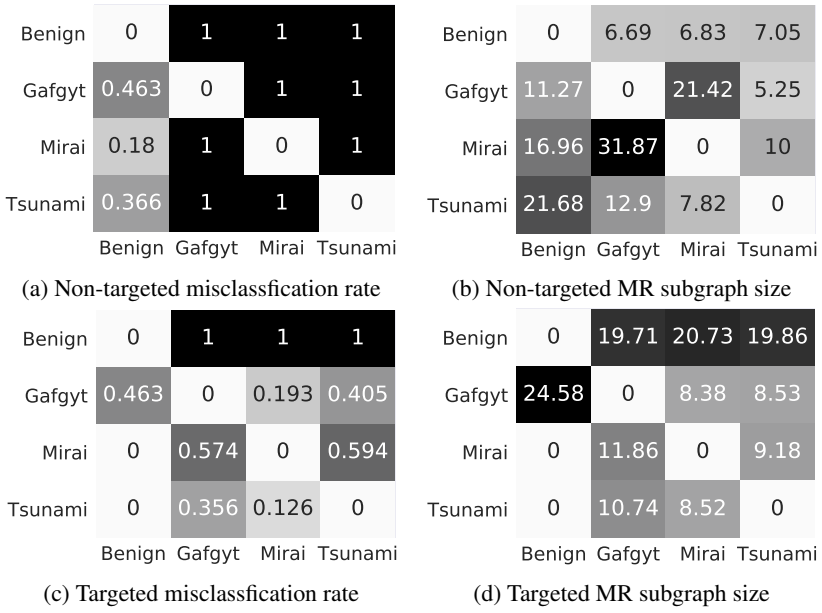
(d) Targeted MR subgraph size

Fig. 7: SGEA: CNN-based IoT malware classification system evaluation. Here, MR refers to misclassification rate, columns represent the sample original class, whereas, rows represent the connected subgraph pattern class.

Summary. The goal of this study is to investigate the robustness of CFG-based IoT malware detection systems against targeted and non-targeted adversarial examples. The main focus of the evaluation is to misclassify the malicious samples into benign. GEA and SGEA have been evaluated over CNN- and DNN-based IoT malware detection systems. In addition, our proposed approach, SGEA, significantly reduces the perturbation overhead of the generated samples, increasing its immunity against detection.

5 Conclusion

In this work, we study the robustness of graph-based deep learning models against adversarial machine learning attacks. To do so, we design SGEA, an approach to generate



(a) Non-targeted misclassfication rate (b) Non-targeted MR subgraph size

(c) Targeted misclassfication rate (d) Targeted MR subgraph size

Fig. 8: SGEA: DNN-based IoT malware classification system evaluation. Here, MR refers to misclassification rate, columns represent the sample original class, whereas, rows represent the connected subgraph pattern class.

AEs corresponding to IoT software by reducing the embedded size to result in model misclassification. The performance of the method is validated through various experiments. We observed that SGEA misclassifies all malware samples as benign while only embedding a subgraph of an average size of 6.8 nodes. This highlights the need for more robust IoT malware detection and classification tools against adversarial learning, particularly those optimized to operate with a small graph perturbation. **Acknowledgement.** This work is supported by NRF grant 2016K1A1A2912757, NVIDIA GPU Grant (2018 and 2019), and a Cyber Florida Seed Grant.

References

1. 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.
2. 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, pp. 88–95, 2019.
3. 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.
4. 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*, 2019.

5. A. Mohaisen, A. Yun, and Y. Kim, "Measuring the mixing time of social graphs," in *ACM IMC*, 2010, pp. 383–389.
6. A. Mohaisen, N. Hopper, and Y. Kim, "Keep your friends close: Incorporating trust into social network-based sybil defenses," in *Proceedings of the 30th IEEE International Conference on Computer Communications, INFOCOM*, 2011, pp. 1943–1951.
7. C. S. Ian J. Goodfellow, Jonathon Shlens, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations.*, 2015, pp. 1–11.
8. 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.
9. 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.
10. K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. D. McDaniel, "Adversarial examples for malware detection," in *ESORICS*, 2017, pp. 62–79.
11. 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.
12. 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.
13. 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.
14. 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.
15. D. Bruschi, L. Martignoni, and M. Monga, "Detecting self-mutating malware using control-flow graph matching," in *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment Conference, DIMVA*, 2006, pp. 129–143.
16. J. Yan, D. Jin, and G. Yan, "Classifying malware represented as control flow graphs using deep graph convolutional neural networks," in *IEEE/IFIP DSN*, 2019, pp. 1–12.
17. 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.
18. 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.
19. A. Khormali, A. Abusnaina, D. Nyang, M. Yuksel, and A. Mohaisen, "Examining the robustness of learning-based ddos detection in software defined networks," in *Proceedings of the IEEE conference on dependable and secure computing, IDSC*, 2019.
20. Developers. (2019) Radare2. Available at [Online]: http://www.radare.org/r/.
21. 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.
22. 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.
23. Developers. (2019) Cyberiocs. Available at [Online]: https://freeiocs.cyberiocs.pro/.
24. ——. (2019) Github. Available at [Online]: https://github.com/.
25. ——. (2019) VirusTotal. Available at [Online]: https://www.virustotal.com.
26. 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.