Unmasking the Lurking: Malicious Behavior Detection for IoT Malware with Multi-label Classification

Ruitao Feng*[†]
rtfeng@smu.edu.sg
Singapore Management University
Singapore

Sen Li[†]
senli@tju.edu.cn
School of Future Technology, Tianjin University
China

Sen Chen
senchen@tju.edu.cn
College of Intelligence and Computing, Tianjin University
China

Mengmeng Ge
mengmeng.ge@ntu.edu.sg
Nanyang Technological University
Singapore

Xuewei Li
lixuewei@tju.edu.cn
College of Intelligence and Computing, Tianjin University
China

Xiaohong Li
xiaohongli@tju.edu.cn
College of Intelligence and Computing, Tianjin University
China

Abstract

Current methods for classifying IoT malware predominantly utilize binary and family classifications. However, these outcomes lack the detailed granularity to describe malicious behavior comprehensively. This limitation poses challenges for security analysts, failing to support further analysis and timely preventive actions. To achieve fine-grained malicious behavior identification in the lurking stage of IoT malware, we propose MaGraMal. This approach, leveraging masked graph representation, supplements traditional classification methodology, empowering analysts with critical insights for rapid responses. Through the empirical study, which took three person-months, we identify and summarize four fine-grained malicious behaviors during the lurking stage, constructing an annotated dataset. Our evaluation of 224 algorithm combinations results in an optimized model for IoT malware, achieving an accuracy of 75.83%. The maximum improvement brought by the hybrid features and graph masking achieves 5% and 4.16%, respectively. The runtime overhead analysis showcases MaGraMal's superiority over the existing dynamic analysis-based detection tool (12x faster). This pioneering work combines machine learning and static features for malicious behavior profiling.

CCS Concepts: • **Security and privacy → Malware and its mitigation**.

*Corresponding author.
[†]Both authors contributed equally.

Keywords: IoT malware, Malicious behavior detection, Multi-label classification, Masked Graph Embedding

ACM Reference Format:
Ruitao Feng, Sen Li, Sen Chen, Mengmeng Ge, Xuewei Li, and Xiaohong Li. 2024. Unmasking the Lurking: Malicious Behavior Detection for IoT Malware with Multi-label Classification. In *Proceedings of the 25th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '24), June 24, 2024, Copenhagen, Denmark.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3652032.3657577

1 Introduction

The exponential growth of the Internet of Things promises transformative impact, with an estimated 100 billion connected devices by 2025 [42]. However, this surge exposes IoT devices to escalating malware threats. Open-source releases like Mirai [5] have spawned advanced variants, including Hajime [15] and Satori [47], resulting in a surge in IoT malware incidents. IoT devices' diverse and resource-constrained nature exacerbates security challenges [9, 11, 20]. Consequently, a mechanism for efficient and precise IoT malware analysis becomes imperative for enhanced protection.

Existing detection methods primarily fall into dynamic and static analysis. While dynamic methods yield detailed insights into malicious behaviors, their resource-intensive and architecture-dependent nature is impractical for IoT devices[10, 12, 24, 45, 46]. Static analysis, on the other hand, is faster but faces challenges in countering emerging malware [2, 23]. Recent advances in machine learning algorithms, utilizing binary file [41, 41, 50], opcode [6, 7, 21, 25], and graph-based features [1, 34, 35, 52, 53], show promise in classification. Traditional methods, often family-focused, lack the precision required for in-depth analysis of behaviors. Consequently, we focus on cross-architecture IoT malware, leveraging Function Call Graphs (FCGs) as the representation in this work. However, FCG analysis reveals redundancy and noise nodes, impacting subsequent fine-grain classification. Unlike binary and family classifications, multi-label classification is more susceptible to noise, emphasizing the need for a refined approach.

In response to the outlined challenges, we refine our research objectives to comprehensively understand fine-grained malicious behaviors in IoT malware that drive actual lurking actions. Specifically, our research draws inspiration from the field of multi-object detection, which aims to locate multiple unrelated targets within an image [8, 26]. Aligning with this, our focus involves utilizing methods from multi-object detection to leverage multi-label classification for a more detailed categorization of IoT malware. This approach assigns multiple labels to each instance [44], facilitating a systematic categorization to identify specific malicious behaviors. This methodology addresses limitations in traditional classification methods, providing richer insights compared to results that often lack valuable clues. As we grapple with the challenge of isolating noise and redundant nodes in the intricate FCG, we draw inspiration from image processing. In image enhancement, masks are often employed to highlight or suppress specific features. Similarly, we introduce a 'masking' strategy for the FCG to identify and suppress irrelevant nodes. By adapting principles, we selectively filter out non-essential elements, refining the FCG into a more focused and meaningful representation. This strategic 'masking' not only streamlines the graph but also significantly improves the precision of our classification results, akin to enhancing the clarity of a digitally processed image.

In this work, we introduce MaGraMal, an innovative approach for multi-label classification of IoT malware. Leveraging a masked graph representation, MaGraMal refines FCG by eliminating redundant and noisy nodes, enabling the identification of fine-grained malicious behaviors during the lurking phase. Through an empirical study, we delineate four behaviors, validated through meticulous manual analysis of 120 representative IoT malware samples. Employing a three-person cross-analysis validation method, we construct a new annotated dataset. Evaluation involves exploring 224 combinations of basic and multi-label classification algorithms, achieving an accuracy rate of 75.83% (logistic+RT). Ablation studies showcase a substantial accuracy improvement in multi-label classification using hybrid features and graph masking compared to those without them (maximum improvement at 5% and 4.16%). Furthermore, detailed runtime performance analysis positions MaGraMal favorably against the existing dynamic analysis-based detection tool (12x faster), highlighting its efficiency.

In summary, we make the following main contributions:

- We propose MaGraMal, a multi-label classification method based on masked graph representation, aimed at identifying fine-grained malicious behaviors of IoT malware in the lurking stage.
- We conduct an empirical study on IoT malware and summarize four fine-grained malicious behaviors in its lurking stage. Based on this, we thoroughly analyze 120 representative samples to construct a new annotated dataset [30].
- We systematically assess the effectiveness of 224 combinations of multi-label classification algorithms on the labeled dataset, identifying the optimal model to serve as a baseline for subsequent evaluations of the soundness of our approach. Simultaneously, we conduct performance comparisons to showcase its superiority over the dynamic analysis-based detection tool.

2 Empirical Study on Lurking Behaviors

In the complex realm of IoT security, understanding fine-grained malicious behavior poses an underexplored challenge. Our empirical study, spanning three person-months, delves into IoT malware to reveal motivations, implementation methods, and execution logic, contributing to a nuanced understanding and taxonomy of lurking behaviors and underlying insight.

2.1 Data Collection and Selection

We sourced 5,000 IoT malware samples from reputable repositories, including VirusShare [48], VirusTotal [49], and IoT-POT [36], spanning 2016 to 2020. Given that 95% of deconstructed firmware in IoT devices is Linux-based [4, 22], our focus lies in analyzing Executable and Linkable Format (ELF) files—ubiquitous in Linux OS. To ensure malware quality, we conducted preprocessing, eliminating non-ELF and corrupted files.

Constructing a large-scale standardized multi-label dataset for IoT malware detection is resource-intensive. Unlike image classification, cybersecurity data annotation necessitates manual analysis by domain experts, consuming significant time and effort. Given these constraints, we opted for a concise yet representative dataset, selecting 120 samples with specific rules. We prioritized representativeness by proportionally selecting from various malicious families, addressing the dominance of *Gafgyt* and *Mirai*(see Appendix § 1.2[29] for details. We also addressed high-similarity issues [43] by choosing one representative sample from similar clusters. These rules streamline manual analysis, ensuring dataset representativeness for robust experimentation.

2.2 Attack Chain Analysis and Behavior Summary

To comprehensively explore the details of each IoT malware sample and ensure no omissions, we conduct a thorough manual analysis of the attack chain of malicious software based on the analysis reports. In this step, we focus on the implementation details, methods, and intentions of malicious behavior in IoT malware, as well as shortcomings in existing technical analysis processes. To achieve complementary advantages from existing techniques and tools, we have devised a multi-level comprehensive IoT malware analysis framework. This framework provides detailed analysis steps and

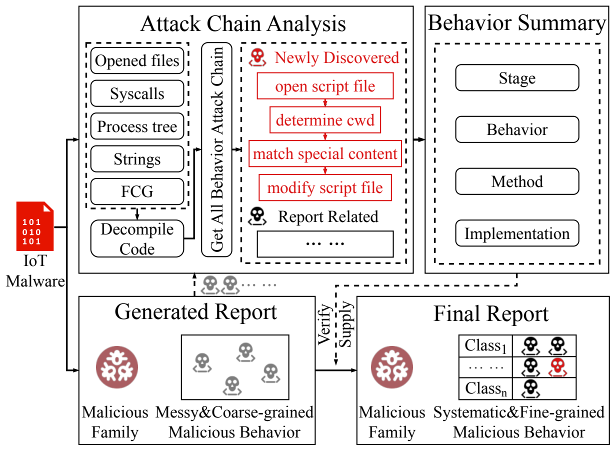


Figure 1. IoT Malware Analysis Process in Our Study

tool recommendations to ensure the comprehensiveness and accuracy of manual analysis. The detailed analysis framework is illustrated in Appendix § 1.1[29]. Specifically, we first upload the samples to VirusTotal for analysis to obtain preliminary analysis reports. Subsequently, as a validation and supplementation step, we individually analyze the selected IoT malware samples using the static analysis tool Ghidra and the dynamic analysis tool LISA. To ensure the credibility of our manual analysis, We referred to the tactics, techniques, and procedures (TTPs) defined in the ATT&CK framework[32] during the analysis process. During this stage, our primary focus is identifying any potentially malicious behaviors that may be overlooked by the reports. As illustrated in Figure 1, the iterative analysis process for each IoT malware sample yields a more comprehensive, detailed, and rigorously validated analysis report. We categorize and summarize fine-grained malicious behaviors based on the conclusive analysis reports. Additionally, it compiles a detailed summary table of malicious behaviors, providing specific details outlined in Appendix § 3[29].

2.3 Insight through Analysis

While analyzing the IoT malware attack chain using FCGs, we observed as we delved deeper into the function call chain of specific malicious behaviors, especially moving away from the FCG center, the analysis required more effort. However, the result of the final analysis indicates that functions represented by these peripheral nodes exhibit a diminishing correlation with the main functionality and core logic of the malicious behavior. After further investigation, we identified that the functions represented by these nodes are typically auxiliary tool functions and standard library functions used in the implementation logic of the attack chain. Importantly, their internal function call relationships, though included in the FCG, are not our focus; we solely rely on their standardized functional descriptions for the main logic of the malicious behavior. Essentially, the function call relationships within these nodes contribute little to our analysis and may even divert our focus negatively. For example, analyzing a standard network programming library function *gethostbyname()* in an IoT malware reveals its main functionality of domain name resolution (i.e., translating hostnames into IP addresses and other information), obviating the need for a deep dive into its internal implementation details (see Appendix § 1.4[29] for details). Consequently, subsequent nodes on this function call chain, farther from the source node, significantly contribute less to our analysis and may even hinder results. In summary, the presence of these peripheral nodes makes analysis complex and may mask key malicious behavior characteristics.

Insight: *While probing specific malicious behaviors in FCGs, peripheral nodes' diminishing relevance, especially distant ones, was observed. A methodology to eliminate such nodes can focus on critical functionalities to better understand malicious behaviors.*

2.4 Definition of Lurking Behaviors

Our aim is to provide security analysts with systematic insights into malicious behaviors through classification results, enabling swift responses. Our strategy prioritizes predicting the granularity of malicious behaviors' semantics over detailed implementation, enhancing generalization for improved detection of new threats. Opting for a higher semantic level reduces computational complexity, ensuring practicality and potential interpretability. By focusing on overall behavior patterns, the model's predictions offer a better understanding for analysts. We analyze IoT malware in three stages: preparation, lurking, and attack, as detailed in Appendix § 3[29].

In the preparation stage, IoT malware engages in activities like gathering target device information, deploying attack tools, and testing methods. Behaviors in this stage may lack distinctiveness, focusing on fine-grained detection might increase false positives, compromising detection credibility. In the attack stage, although behaviors are distinct, their direct and simple interaction with IoT device systems limits their preventive fortification. Thus, more effective prevention and detection should be adopted upon the lurking stage. Behaviors here possess significant distinction, as their implementation relies more on the target device, requiring profound interaction with system resources. More importantly, detecting fine-grained malicious behaviors in the lurking stage allows security analysts to take more effective defensive measures, aiming to hinder potential substantive attack behaviors at an early stage [3].

Following an in-depth manual analysis aligned with the tactics defined in MITRE ATT&CK framework, we distill our findings into four distinct fine-grained malicious behaviors occurring in the lurking stage of IoT malware. Correspondingly, we coin the labels: **Persistence**, **Privilege Escalation**, **Deception**, and **Defense Evasion**. These behaviors directly

Table 1. Label Distribution of Annotated Dataset

Label	Description	#
Persistence	Maintaining long-term control over the victim's system after restart.	88
Privilege Escalation	Acquiring higher system permissions for more destructive or theft activities.	28
Deception	Hiding its true intention or identity by disguising as or simulating normal applications, thereby deceiving users or defense systems.	69
Defense Evasion	Evading security software to ensuring malware not being detected and blocked in the victim's system.	102

impact the malware's survivability, attack potency, and ability to remain concealed—critical factors in executing lurking intents by existing IoT malware. An overview and implementation mechanism of these behaviors can be found in the Appendix § 2 [29]. Table 1 illustrates the distribution of these malicious behaviors in our constructed dataset.

2.5 Data Annotation

After comprehensively analyzing IoT malware attack chains, we annotate each sample using a four-dimensional vector representing lurking behaviors. Each dimension in the vector is marked as "1" if the corresponding behavior is present and "0" if absent. For dataset annotation quality control, we adopted a cross-analysis verification method involving three individuals in this work to ensure the high accuracy and reliability of the data annotation. This quality control step is crucial for ensuring the credibility of our research results. In this process, two annotators conduct independent analyses and annotations of the IoT malware, while a third individual is responsible for verifying and confirming any disputed analysis results. Through this three-person cross-analysis verification phase, we can eliminate potential annotation errors and inconsistencies, thus ensuring our research results are reliable and highly accurate.

3 Approach

3.1 Overview

The proposed *MaGraMal* workflow, depicted in Figure 2, consists of four key phases:

- **Graph Representation**: We perform static analysis on the input cross-architecture IoT malware binary files using Radare2 [38] and construct the corresponding FCG.
- **Masked Graph Construction**: To determine the masking area, we leverage centrality analysis to pinpoint central nodes and fine-tune the balance parameter α to optimize the mask radius, enabling the diverse masking levels.
- **Feature Extraction**: Feature extraction from the masked graph concentrates on structural and embedding features, incorporating global structural information and local node semantics, thereby lessening information loss.
- **MLC Model Construction**: 224 distinct multi-label classification models are constructed using various basic and multi-label classification algorithms.

3.2 Graph Representation

This phase focuses on constructing graphical representations for Cross-Architecture IoT malware. Graph-based methods, proven effective by existing work [27] and our analysis of real-world samples, adeptly capture intricate relationships, resist code obfuscation, and accommodate the differential characteristics of Cross-Architecture IoT malware. Malware compiled on diverse CPU architectures exhibits variations due to distinct instruction sets. Despite these differences, leveraging FCG empowers machine learning algorithms to identify similarities at the function call level. This approach addresses the challenge of cross-architecture differences in learning, facilitating knowledge transfer across diverse subdomains of malware detection. We utilize Radare2 for static analysis, generating a concise FCG from a Control Flow Graph (CFG). While the CFG accurately captures interactions during execution, it tends to produce significantly large graphs, even for moderately-sized binary files, leading to a substantial increase in processing and analysis time. In contrast, the FCG provides a more streamlined representation, prioritizing performance and graphical clarity, consequently, adopted as our representation in this work.

3.3 Masked Graph Construction

When analyzing and summarizing the attack chains of IoT malware, we observed that certain function calls presented in the FCG did not provide useful information for our analysis. With curiosity in mind, we delved into the underlying implementation of key malicious behaviors by tracing the function calls. An interesting observation reveals that the implementation of such behaviors always ends with nodes representing auxiliary tool functions or standard library functions. Therefore, we only need to focus on the functionality they provide, without concerning ourselves with their internal implementation details. Essentially, their internal implementation details contribution to comprehending malicious behavior is negligible, leading us to question whether they can add positive value during machine learning. This question will be answered with the results of an ablation study presented in § 4.4.2.

In other words, within an FCG, surrounding nodes closer to and execute prior to such function calls typically provide more valuable information. Conversely, nodes far from the source offer limited value and even potentially introduce substantial redundancy and noise during machine learning. This dynamic can obscure crucial features of malicious behavior, introducing complexity and uncertainty into machine learning and impacting model efficiency and accuracy. Therefore, we introduce the Masked Graph, a more precise and clean malware representation, achieved by eliminating the aforementioned peripheral nodes according to specific domain knowledge obtained by our empirical study (details in § 2.3). This approach significantly reduces the redundancy

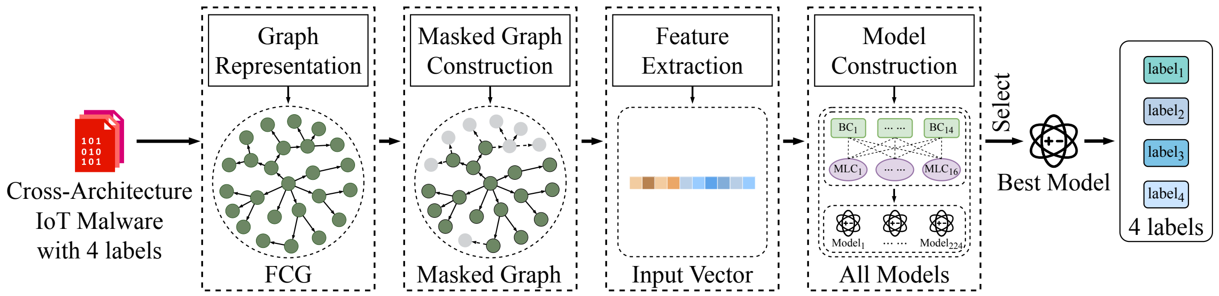


Figure 2. The Overview of MaGraMal

and noise impact caused by peripheral nodes according to our evaluation results in § 4.4.2.

3.3.1 Key Components of Masked Graph. Before introducing the approach, we establish definitions for key components essential to understanding masked graph construction.

Definition 1 (*Central Node*)**.** *The central node, typically the program's entry point, holds a crucial role in the FCG, linking program segments and dictating calling relationships. Analyzing it and its surroundings unveils primary functions and behaviors, offering vital cues for subsequent manual or automated analysis. Key central node characteristics encompass:*
Centrality. *Typically located in the core area, central nodes are associated with multiple core functionalities, enhancing their prominence in the graph.*
High Degree of Connectivity. *Central nodes in the FCG or graph generally have many connecting edges, indicating close associations and interactions with other nodes.*
Wide-ranging Influence. *Central nodes typically exert significant influence with their behavior or attributes significantly implicating the operation of the entire program.*

Definition 2 (*Edge Node*)**.** *Edge nodes, positioned at the periphery of an FCG, signify functions near the program's end, often linked to auxiliary or library functions. These nodes typically offer limited insights into malware core logic, introducing redundancy and noise. Key characteristics include:*
Marginality. *Edge nodes are usually located at the edges of an FCG, distinct from the center and unrelated to the core behaviors.*
Sparse Connectivity. *Edge nodes typically exhibit fewer connections to other nodes in the graph, earning them the designation of "fringe members". Unlike central nodes, they are not as densely connected within the graph.*
Local Influence. *Edge nodes' role may be limited to specific local areas of the graph, resulting in minimal global impact on program semantics.*

Definition 3 (*Mask Radius*)**.** *The mask radius in graph masking sets the distance threshold from the central node to the covered edge nodes. Beyond the radius, nodes are labeled edge nodes; within, they are not. This parameter's value impacts the tradeoff between preserving semantic information and graph size reduction. The choice hinges on analysis needs and masking objectives.*

3.3.2 Mask Radius Calculation. Mask radius calculation faces a challenge with graphs of varying sizes and characteristics in FCGs. The complexity of IoT malware implementations results in significant variations in the number of nodes in their FCGs. A fixed mask radius is impractical, risking over-masking complex graphs and losing valuable information, or under-masking simpler graphs and retaining redundant nodes. To address this, we propose an adaptive mask radius calculation method that dynamically adjusts the mask radius based on the graph's scale for more precise and effective masking tailored to diverse situations.

To gain an approximate scale and outline of the FCG, we first create a set $Set_{spath}(G, v_{center})$, containing the shortest paths from the central node v_{center} to all other reachable nodes v_m in graph G. Next, we proceed with the calculation of the mask radius. The calculation involves two key factors: the balance parameter α and the maximum value in the set $Set_{spath}(G, v_{center})$. Specifically, to adaptively determine the mask radius size for FCGs of different scales, we select the maximum value in $Set_{spath}(G, v_{center})$ as the base radius. Introducing a balance parameter α controls the final mask radius size, striking a balance between retaining essential information and eliminating redundant and noisy nodes. This approach tailors the mask radius to the specific characteristics of each graph. Finally, the calculated result of $\alpha \cdot base\_radius$ is rounded up to obtain the optimal mask radius. Through this adaptive method, we can flexibly fine-tune the size of the mask radius, adapting to the various characteristics of graphs.[1].

Giving a malware binary P, its FCG is represented as a directed graph $G = \{V, E\}$, where $V = \{v_1, v_2, v_3, ..., v_m\}$ is the set of vertices, and $E = \{(e_{i,j})\}$ is the set of directed edges. The calculation of the adaptive mask radius is defined as follows:

$$\begin{cases} Set_{spath}(G, v_{center}) = \{ShortestPath_{\forall center \neq m}(v_{center}, v_m)\} \\ base\_radius = Max(Set_{spath}(G, v_{center})) \\ mask\_radius = \lceil \alpha \cdot base\_radius \rceil, \ \alpha \in (0, 1] \end{cases}$$

where $ShortestPath(v_{center}, v_m)$ is the shortest path between the central node and node i.

[1]The effectiveness is validated in mitigating the impact of noise in lurking behavior detection on real-world data. Discussion on potential threats brought by adversarial attacks can be found in Appendix § 1.3[29]

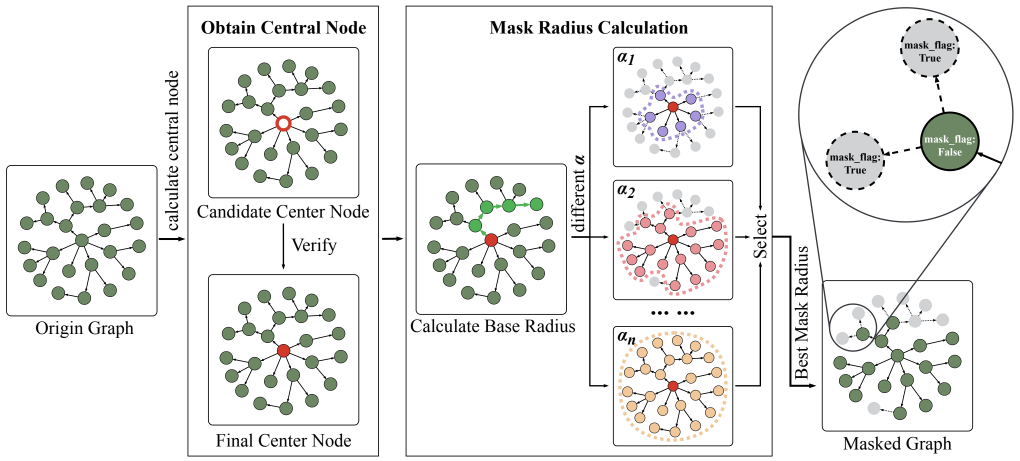


Figure 3. Masked Graph Construction Process

3.3.3 Graph Masking Process. We elaborate on the process of graph masking in Figure. 3, where the input is the FCG of IoT malware. The output is a masked graph following the outlined masking procedures as follows:

Step 1. Our first step is to acquire the central node. For IoT malware, given that the execution typically begins from the program entry function, we logically designate it as the candidate central node temporarily. Note that, due to the possibility of certain malicious software employing multi-layered function nesting to conceal critical malicious behaviors, this may result in the significant departure of the implementation nodes of such key malicious behaviors from the program entry function node. If, at this point, we directly define the entry function node as the central node, critical malicious behaviors implementation nodes will be considered as noise and consequently masked. Hence, we introduce degree centrality to validate the candidate central node obtained above. We treat the FCG as a social network and use degree centrality to measure the "social connections" (paths) of each "individual" (node). Central nodes resemble individuals with extensive social connections in a social network, exhibiting high centrality and many dense function calls, setting them apart from other nodes. Specifically, we calculate the degree centrality $C_{degree}(v_i)$ for all nodes in the graph and rank them from highest to lowest. The mathematical definition of degree centrality is as follows:

$$C_{degree}(v_i) = \frac{out\_degree(v_i)}{|V| - 1}$$

where $out\_degree(v_i)$ is the out degree of node v_i, which is the number of outgoing edges from node v_i, and V represents the set of all nodes.

Subsequently, we assessed whether the degree centrality of the candidate central node, determined based on the program entry function, ranks among the top. This validation ensures that the candidate holds a central position by comparing its centrality to others. Therefore, if no other nodes have a higher degree of centrality with practical analytical value, the candidate is considered the central node. Otherwise, we select the node with the highest degree of centrality and significant practical relevance.

Step 2. Based on the central node identified in the previous step, and using the mask radius calculation method introduced in § 3.3.2, we calculate the base radius and mask radius for the FCG. In this step, we can adjust the balance parameter α to find an optimal balance point that preserves key semantic information in the graph while maximally eliminating redundant and noisy nodes.

Step 3. We set a mask flag for each node, indicating whether the node will be filtered out in the subsequent graph analysis process. It can be calculated by:

$$\begin{cases} distance = ShortestPath(v_{center}, v_i) \\ mask\_flag_i = \begin{cases} true, \ distance > mask\_radius \\ false, \ distance \leq mask\_radius \end{cases} \end{cases}$$

The specific approach involves calculating the shortest path between each node and the central node, comparing it to the mask radius. If the distance exceeds the radius, the node is marked as an edge node and flagged for masking; conversely, if it is set as unmasked. Eventually, this process yields the final masked graph. Subsequent analysis filters nodes based on the mask flags. This approach effectively removes redundant and noisy nodes, improving the multi-label classification's effectiveness and significantly reducing the graph size.

3.4 Feature Extraction

The feature extraction step transforms the masked graph into a numerical vector representation. Graph embedding, achieved through graph2vec [33], efficiently maps high-dim graph data to low-dim dense vectors, capturing latent patterns and semantic information effectively. This unsupervised representation learning technique outperforms traditional algorithms in scalability and computational adaptability [51]. To overcome graph embedding feature limitations, we incorporate global insights by introducing structural attributes such as node count (indicating identified functions), edge count (illustrating function call relationships), and graph density (depicting dependency and interaction patterns). These attributes provide a comprehensive view of the function call graph's overall structure with minimal performance overhead, enhancing our understanding of the binary file. The mathematical definition of graph density is as follows:

$$Density(G) = \frac{\sum_{i=1}^{m} deg(v_i)}{m(m-1)}$$

where $deg(v_i)$ is the degree of node i, $V = v_1, v_2, v_3, ..., v_m$. In conclusion, the input for subsequent models is a $(D+3)$-dimensional numerical vector combining graph embedding (D-*dimension*) and structure features (3-*dimension*).

3.5 MLC Model Construction

We will primarily discuss the construction of the multi-label classification model for IoT malware. We first adopt the graph embedding features and graph structural features extracted in § 3.4 as a unified input vector. The paired input vectors and the labels representing fine-grained malicious behaviors for each sample are fed into the multi-label classification algorithms. In our work, a total number of 224 combinations of machine learning algorithms for muli-label detection is selected. Specifically, we employ 14 basic classification algorithms, whose categories and brief descriptions are shown in Appendix § 3[29]. Additionally, based on these basic classification algorithms, we utilize 16 multi-label classification algorithms. These algorithms fall into two categories of approaches: algorithm adaptation [44] and problem transformation [13]. Algorithm adaptation involves extending traditional single-label classification algorithms to multi-label classification tasks to enhance performance and adaptability. It aims to adapt single-label classification algorithms to multi-label contexts, often by altering the output or loss function of the single-label classification algorithm. Problem transformation converts multi-label problems into a series of single-label problems, which are then addressed using single-label classifiers. This method is flexible as it can use any single-label classifier. The correspondence between these multi-label classification algorithms and the two methods are shown in Appendix § 3[29], with 2 belonging to the algorithm adaptation method and 14 to the problem transformation method. We ultimately identified the best-performing classifier by comparing the performance of those above 224 multi-label classification models.

4 Evaluation

We evaluate MaGraMal by addressing the following RQ:

RQ1: How is the effectiveness of using multi-label classification in IoT malware detection? This RQ aims to compare the results of different multi-label classification models under various balance parameter α values to validate the effectiveness of MaGraMal in classifying malicious behaviors during the lurking phase of IoT malware.

RQ2: How is the soundness of crucial design? This RQ focuses on ablation experiments targeting graph structural features and graph masking to validate soundness.

RQ3: How is the performance of MaGraMal in detecting IoT malware? This RQ explores the runtime overhead of MaGraMal and compares it with an IoT malware sandbox to investigate performance differences.

4.1 Experimental Environment

All experiments are conducted on the Ubuntu server of Intel(R) Xeon(R) E5-2620 v4 and 36GB RAM and implemented on MEKA v1.9.2 [39] [31] in Python 3.8.

4.2 Datasets and Metrics

The dataset comprises 120 representative IoT malware samples selected from a preliminary analysis of 5,000 samples, labeled based on the four lurking stages of malicious behavior (see § 2.4 and Table 1 for details). MaGraMal evaluation involves 5-fold cross-validation, dividing the dataset into subsets. Each iteration designates one subset as the test set, while the remaining 4 form the training set. This process iterates 5 times, and average values constitute the final results.

Evaluation metrics, following Q. Qiao et al. [37], include Sample-ACC, F1-Score, Hamming Loss, Zero-One Loss for sample-based assessment, and Label-ACC for label-based assessment, collectively gauging MaGraMal's effectiveness.

4.3 RQ1: Effectiveness

4.3.1 Different Multi-Classification Models.

Setup. We experimented on 224 multi-label classification models, including 14 basic and 16 multi-label algorithms, in the evaluation of effectiveness. Employing 5-fold cross-validation enabled a comprehensive and robust assessment, identifying optimal algorithmic combinations.

Result. In Table 2, we present experimental results, ranking models by sample accuracy. Optimal results are highlighted in bold with a gray background for emphasis.

For sample accuracy, Logistic Regression combined with Random Tree achieved the highest at 0.7167 among the 224 models. Models 2 to 6 closely followed with a 0.7 accuracy, while Models 7 and 8 scored 0.6917. Notably, the Random Forest and RAkEL model, not the highest in sample accuracy, excelled in Hamming loss (0.1833), Zero-One loss (0.4583), and F1-Score (0.8429). This signifies its effectiveness in minimizing label prediction mismatches, maintaining a balance between precision and recall. Beyond sample accuracy, label accuracy ($Label\text{-}ACC_{avg}$) was computed for each model, representing the average accuracy of the four labels. Model 2, second in sample accuracy, led in label accuracy with 0.8167, showcasing its precision in predicting individual labels.

Table 2. The Results of Top 8 Models with Sample-ACC

Model Rank	1	2	3	4	5	6	7	8
BC	Logistic	RF	Logistic	Logistic	Logistic	SMO	RF	Logistic
MLC	RT	RAkEL	BCC	CT	HASEL	CDN	BR	BR
Hamming loss	0.2042	**0.1833**	0.2021	0.2000	0.1854	0.2146	0.1979	0.2042
Zero-One loss	0.5333	**0.4583**	0.5167	0.5000	0.4917	0.5000	0.5250	0.5333
F1-score	0.8176	**0.8429**	0.8248	0.8268	0.8399	0.8129	0.8288	0.8229
Label$_1$-ACC	0.7417	**0.8000**	0.7917	0.7917	0.7917	0.7500	0.7917	0.7833
Label$_2$-ACC	**0.8750**	0.8250	0.8333	0.8333	0.8583	0.7833	0.8333	0.8167
Label$_3$-ACC	0.7833	0.7917	0.8250	0.8167	**0.8333**	0.7583	0.7500	0.7917
Label$_4$-ACC	0.7833	**0.8500**	0.7417	0.7583	0.7750	**0.8500**	0.8333	0.7917
Label-ACC$_{avg}$	0.7958	**0.8167**	0.7979	0.8000	0.8146	0.7854	0.8021	0.7958
Sample-ACC	**0.7167**	0.7000	0.7000	0.7000	0.7000	0.7000	0.6917	0.6917

From 224 models, the top 8 were identified, with optimal sample and label accuracies reaching 0.7167 and 0.8167, affirming the viability and effectiveness of multi-label classification in IoT malware detection.

4.3.2 Different Balance Parameter α.

Setup. We utilized the top-performing 8 models from Experiment § 4.3.1. Assessing sample-based accuracy as the primary metric, we investigated the impact of different balance parameters (α) on classification. Considering the diverse base radius of function call graphs in IoT malware (typically 4 to 9), we selected five α values (0.1, 0.3, 0.5, 0.7, 0.9) to avoid unnecessary resource consumption.

Result. Figure 4 depicts how different models' accuracy varies with the balance parameter α. The x-axis represents α, and the y-axis shows sample accuracy. Analysis reveals fluctuations in accuracy across the eight models, indicating the significant impact of different α values on masked graph construction and subsequent classification.

Upon in-depth analysis, we find that when α is 0.7, each model achieves its highest or tied-for-highest accuracy (Figure 4a to Figure 4f). Models 7 and 8 also exhibit tied-for-highest accuracy at α=0.7 (Figure 4g and Figure 4h). This suggests that at α=0.7, the masked graph effectively filters out redundant and noisy nodes, optimizing the balance between information retention and noise reduction, thereby enhancing classification capabilities.

4.4 RQ2: Soundness

4.4.1 Ablation for Graph Structural Feature.

Setup. Leveraging the 8 optimal models from § 4.3.1, we employed two input construction methods. The first utilized only graph embedding features for model training. The second incorporated both graph embedding and structure features. Comparing the classification results of models trained with these methods assesses the impact of adding graph structure features on classification enhancement.

Result. In Table 3, we showcase model results utilizing two distinct input vector types. Shaded entries signify models trained with graph structure features, differentiating them from those relying solely on graph embedding features. Blue arrows highlight trends for models using both features: upward denotes an increase, while downward signals a decrease. Hamming Loss and Zero-One Loss reductions indicate enhanced classification, whereas F1-Score, Label-ACC, and Sample-ACC increments signify improved capabilities.

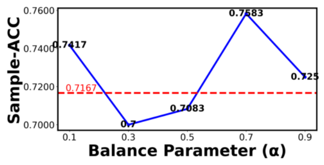
Across the 8 best models, Models 1 to 5 show significant enhancement in all metrics with added graph structure features. Models 6 and 8 exhibit optimization, while Model 7 shows a slight decline. 6 models show significant gains in Sample-ACC, with Model 4 leading by 0.0667. Even the smallest improvement is 0.0083, while Model 6 remains stable and Model 7 experiences a slight drop. In terms of Label-ACC, 5 models demonstrate marked improvement, with Model 1 leading at 0.0437. Model 8 maintains stable accuracy. Notably, Model 2 achieves a 0.8429 F1-Score, and Model 1 shows the highest improvement at 0.0409. After incorporating graph structure features, 7 models exhibit substantial F1-Score improvement, indicating a refined balance between precision and recall—critical for practical applications. Most models experience varying degrees of enhancement in Hamming Loss and Zero-One Loss, suggesting an improved understanding of intricate data relationships and better label differentiation.

In summary, introducing graph structural features during feature extraction benefits 7 out of 8 models, enhancing F1-Score and Zero-One Loss. Top improvements reach 0.0409 in F1-Score and a 0.0917 reduction in Zero-One Loss. Six models either improve or maintain performance across sample accuracy, label accuracy, and Hamming Loss. Variability in utilizing structural features arises due to algorithmic constraints or model complexity, yielding inconsistent improvements across models and metrics. Nonetheless, results affirm that most models significantly benefit from incorporating graph structural features.

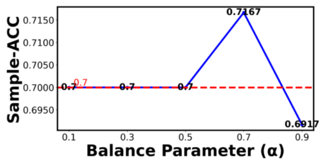
4.4.2 Ablation for Masked Graph.

Setup. To gauge the enhancement with the masked graph, we conducted comparative experiments with and without the graph masking on the top 8 models with optimal α=0.7.

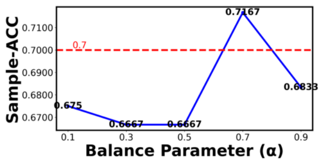
Result. In Figure 4, we introduced a red dashed line parallel to the x-axis as a reference for the model's accuracy without the masked graph, enhancing readability. The blue curve represents the accuracy variation after introducing



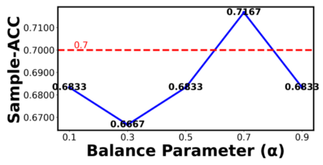
(a) Logistic&RT



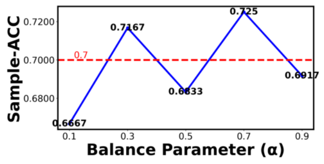
(b) RF&RAkEL



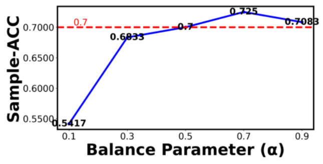
(c) Logistic&BCC



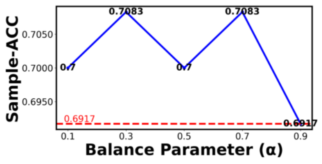
(d) Logistic&CT



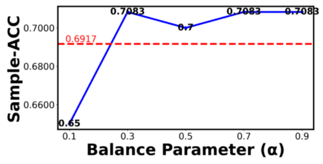
(e) Logistic&HASEL



(f) SMO&CDN



(g) RF&BR



(h) Logistic&BR

Figure 4. The Plots of Sample-ACC on Different α

Table 3. The Results of Top 8 Models with Sample-ACC

Model	1	2	3	4	5	6	7	8
BC	Logistic	RF	Logistic	Logistic	Logistic	SMO	RF	Logistic
MLC	RT	RAkEL	BCC	CT	HASEL	CDN	BR	BR
Hamming loss$_1$	0.2479	0.1896	0.2042	0.2208	0.2042	0.2083	0.1833	0.2042
Hamming loss	0.2042↓	0.1833↓	0.2021↓	0.2000↓	0.1854↓	0.2146	0.1979	0.2042
Zero-One loss$_1$	0.6250	0.5000	0.5417	0.5750	0.5583	0.5167	0.4917	0.5833
Zero-One loss	0.5333↓	0.4583↓	0.5167↓	0.5000↓	0.4917↓	0.5000↓	0.5250	0.5333↓
F1-score$_1$	0.7758	0.8392	0.8185	0.8047	0.8216	0.8201	0.8415	0.8208
F1-score	0.8176↑	0.8429↑	0.8248↑	0.8268↑	0.8399↑	0.8129↑	0.8288	0.8229↑
Label-ACC$_1$	0.7521	0.8104	0.7958	0.7792	0.7958	0.7917	0.8167	0.7958
Label-ACC	0.7958↑	0.8167↑	0.7979↑	0.8000↑	0.8146↑	0.7854	0.8021	0.7958
Sample-ACC$_1$	0.6667	0.6917	0.6667	0.6333	0.6583	0.7000	0.7167	0.6583
Sample-ACC	0.7167↑	0.7000↑	0.7000↑	0.7000↑	0.7000↑	0.7000	0.6917	0.6917↑

the masked graph. Experiment § 4.3.2 revealed the optimal result at $\alpha = 0.7$. In Figure 4, under this value, the model consistently outperforms the one without the masked graph, maintaining at least the same level even in the worst-case scenario. For Model 1, sample accuracy increased from 0.7167 to 0.7583, a significant improvement of 0.0416. Similarly, Models 7 and 8 saw an increase from 0.6917 to 0.7083, representing an improvement of 0.0166, though modest compared to other models. Based on this, we can answer the questions in § 3.3. Notably, Certain α values may lead to slightly lower sample accuracy than the model without the masked graph, suggesting over-masking. This occurs when mask operations with a specific α are applied to the FCG, potentially masking nodes containing crucial semantic information and degrading the model's classification ability.

4.5 RQ3: Performance

4.5.1 Runtime Overhead.

Setup. We deployed the best models from Experiment § 4.3.1, conducting end-to-end analysis on 120 samples. We monitored system performance metrics, to explore MaGraMal's analysis time composition and its overall performance.

Result. In the detection process of MaGraMal, illustrated in Figure 5, three distinct stages delineate the runtime overhead, each marked by different background colors:

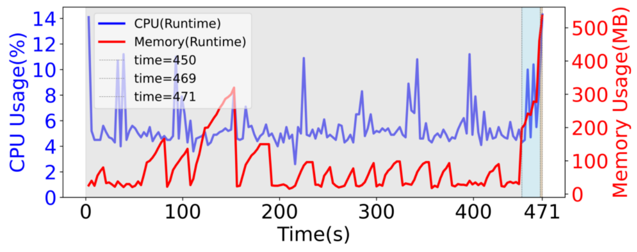


Figure 5. Runtime Performance of MaGraMal

(1) FCG Extraction: MaGraMal initiates by extracting the function call graph of the given IoT malware. As shown in Figure 5, this stage consumes the majority of runtime resources, constituting nearly 95% of the overall analysis time. Notably, peaks in memory usage correlate with larger IoT malware samples processed by Radare2.

(2) Feature Extraction: This stage involves constructing a masked graph based on the function call graph and extracting graph structure and embedding features. As shown in Figure 5, it takes approximately 19 seconds, with CPU and memory usage exhibiting acceptable increments.

(3) Model Classification: In the final stage (i.e., prediction), despite increased CPU and memory usage, the completion time is merely about 2 seconds, contributing a minimal 0.4% to the overall analysis time. This stage occupies the smallest runtime overhead among the three steps.

Table 4. Performance Compared to LISA [45]

System	CPU Usage (%)	Memory Usage (MB)	Execution Time (second)
MaGraMal	$P_{6\%}$	83	4
LISA [45]	$P_{15\%}$	80	51

4.5.2 Comparison with Sandbox.

Setup. To assess the runtime performance difference between our MaGraMal and existing fine-grained behavior analysis tools, we chose LISA[45], a sandbox developed specifically for dynamic analysis of IoT malware. Although LISA's reports do not directly offer behavior analysis report, their extensive analysis logs are sufficient to provide key clues for subsequent in-depth analysis. Similarly, our MaGraMal, by revealing lurking behaviors without meticulous analysis, also serves as a foundation for subsequent in-depth analysis, offering crucial clues. In essence, our objective aligns: to furnish key clues for subsequent analyses, thereby ensuring the comparability and completeness of the analysis.

Result. The performance metrics in Table 4 highlight the resource consumption during the runtime of the tools. MaGraMal exhibits superior efficiency with a CPU usage of 6%, outperforming LISA at 15%. While memory consumption is comparable (83MB for MaGraMal and 80MB for LISA), MaGraMal excels in execution time, averaging 4 seconds compared to LISA's 51 seconds. The results demonstrate that our system outperforms LISA Sandbox in overall performance.

5 Related Work

With the rising tide of malware, IoT malware classification becomes increasingly crucial, compared to a broader scope [16–19, 37]. IoT Malware classification can entail categorizing binary files as malicious or benign, or further classifying malware samples into different known malicious software families. Existing methods are mainly divided into two categories: dynamic methods [10, 12, 28, 45, 46] and static methods[2, 23]. Dynamic methods refer to researchers dynamically executing malware in a virtual environment to simulate its real behavior, enabling them to collect useful information such as API call sequences. Jeon et al.[24] proposed a method to dynamically analyze IoT malware on an ARM architecture virtual machine, extract relevant behavioral features transformed into images, and classify them using CNN. On the other hand, static methods refer to the static inspection of software code to identify malicious code within it. Compared to dynamic methods, static methods are faster and easier to implement. Alhanahnah et al.[2] proposed a method for coarse-grained clustering by generating signatures based on statistical data and printable string features. They also utilized bindiff[54] to calculate the similarity of binary file structures for fine-grained clustering. With the proliferation of machine learning, researchers have begun to utilize machine learning for IoT malware classification[37]. Depending on the features used in the research, related work can be categorized into binary-based[41, 50], opcode-based[6, 21, 25], graph-based[1, 35, 52], etc. Deng et al.[14] proposed a method to generate an ASCG by improving the semantic representation of dynamic system call information using information from the Linux Programmer's Manual. Ban et al.[7] proposed a method for characterizing IoT malware using multimodal analysis. Nguyen et al.[34] proposed a method for extracting information from printable strings presented in FCG to generate PSI-Graph.

6 Discussion and Future Work

While our method effectively tackles redundancy and noise in covert behavior detection, it's a preliminary solution focused on efficient lurking stage malicious behavior detection. However, limitations persist: (1) Similar to existing ML/DL-based malware detection models [40], adversarial attacks introducing extreme-scale samples may challenge the existing masked graph approach, affecting performance (see Appendix § 1.3[29] for details); (2) The current method lacks precision in distinguishing crucial information and redundancy within the same mask radius, leading to potential over- or under-masking; (3) A uniform alpha setting may not consider individual differences between samples, impacting classification; (4) Models trained on limited data may struggle with emerging and evolving fine-grained malicious behaviors, limiting generalization. To address these limitations, future research will focus on: (1) Improving adaptability to extreme-scale samples for enhanced resilience against adversarial attacks; (2) More efficient mechanisms to pinpoint crucial information in the function call graph; (3) Exploring self-adaptive alpha adjustment strategies to different malware; (4) Updating mechanism for new implementation methods of lurking behaviors in recent IoT malware.

7 Conclusion

This paper presents MaGraMal, a novel method designed for multi-label classification of IoT malware. The approach is based on a masked graph representation, which refines the FCG by identifying and suppressing redundant and noisy nodes, thereby producing more targeted and meaningful representation. Through this methodology, we successfully identify and characterize four fine-grained malicious behaviors exhibited by IoT malware during the lurking stage.

Acknowledgments

This work was partially supported by the National Key R&D Project (2021YFF1201102), the National Natural Science Foundation of China (Key Program , Grant No.62332005, Grant No.62102284), the Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN), and the Smart Platform Infrastructure Research on Integrative Technology (SPIRIT) under Grant CSA-CSEC-DC-20-083.

References

[1] Hisham Alasmary, Aminollah Khormali, Afsah Anwar, Jeman Park, Jinchun Choi, Ahmed Abusnaina, Amro Awad, Daehun Nyang, and Aziz Mohaisen. 2019. Analyzing and detecting emerging Internet of Things malware: A graph-based approach. *IEEE Internet of Things Journal* 6, 5 (2019), 8977–8988.

[2] Mohannad Alhanahnah, Qicheng Lin, Qiben Yan, Ning Zhang, and Zhenxiang Chen. 2018. Efficient signature generation for classifying cross-architecture IoT malware. In *2018 IEEE conference on communications and network security (CNS)*. IEEE, 1–9.

[3] Omar Alrawi, Charles Lever, Kevin Valakuzhy, Kevin Snow, Fabian Monrose, Manos Antonakakis, et al. 2021. The Circle of life: A {large-scale} study of the {IoT} malware lifecycle. In *30th USENIX Security Symposium (USENIX Security 21)*. 3505–3522.

[4] Kishore Angrishi. 2017. Turning internet of things (iot) into internet of vulnerabilities (iov): Iot botnets. *arXiv preprint arXiv:1702.03681* (2017).

[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the mirai botnet. In *26th USENIX security symposium (USENIX Security 17)*. 1093–1110.

[6] Amin Azmoodeh, Ali Dehghantanha, and Kim-Kwang Raymond Choo. 2018. Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. *IEEE transactions on sustainable computing* 4, 1 (2018), 88–95.

[7] Tao Ban, Ryoichi Isawa, Shin-Ying Huang, Katsunari Yoshioka, and Daisuke Inoue. 2019. A cross-platform study on emerging malicious programs targeting iot devices. *IEICE TRANSACTIONS on Information and Systems* 102, 9 (2019), 1683–1685.

[8] Matthew R Boutell, Jiebo Luo, Xipeng Shen, and Christopher M Brown. 2004. Learning multi-label scene classification. *Pattern recognition* 37, 9 (2004), 1757–1771.

[9] Rajasekhar Chaganti, Vinayakumar Ravi, and Tuan D Pham. 2022. Deep learning based cross architecture internet of things malware detection and classification. *Computers & Security* 120 (2022), 102779.

[10] Cheng-Yu Chen and Shun-Wen Hsiao. 2019. IoT malware dynamic analysis profiling system and family behavior analysis. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 6013–6015.

[11] Andrei Costin and Jonas Zaddach. 2018. Iot malware: Comprehensive survey, analysis framework and case studies. *BlackHat USA* 1, 1 (2018), 1–9.

[12] Ahmad Darki and Michalis Faloutsos. 2020. RIoTMAN: a systematic analysis of IoT malware behavior. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*. 169–182.

[13] Alex GC de Sá, Cristiano G Pimenta, Gisele L Pappa, and Alex A Freitas. 2020. A robust experimental evaluation of automated multi-label classification methods. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 175–183.

[14] Liting Deng, Hui Wen, Mingfeng Xin, Hong Li, Zhiwen Pan, and Limin Sun. 2023. Enimanal: Augmented cross-architecture IoT malware analysis using graph neural networks. *Computers & Security* 132 (2023), 103323.

[15] Sam Edwards and Ioannis Profetis. 2016. Hajime: Analysis of a decentralized internet worm for IoT devices. *Rapidity Networks* 16 (2016), 1–18.

[16] Ruitao Feng, Sen Chen, Xiaofei Xie, Lei Ma, Guozhu Meng, Yang Liu, and Shang-Wei Lin. 2019. MobiDroid: A Performance-Sensitive Malware Detection System on Mobile Platform. In *2019 24th International Conference on Engineering of Complex Computer Systems*.

[17] Ruitao Feng, Sen Chen, Xiaofei Xie, Guozhu Meng, Shang-Wei Lin, and Yang Liu. 2020. A Performance-Sensitive Malware Detection System Using Deep Learning on Mobile Devices. *IEEE Transactions on Information Forensics and Security* (2020).

[18] Ruitao Feng, Jing Qiang Lim, Sen Chen, Shang-Wei Lin, and Yang Liu. 2020. SeqMobile: An Efficient Sequence-Based Malware Detection System Using RNN on Mobile Devices. In *2020 25th International Conference on Engineering of Complex Computer Systems*.

[19] Ruitao Feng, Yang Liu, and Shangwei Lin. 2019. A Performance-Sensitive Malware Detection System on Mobile Platform. In *Formal Methods and Software Engineering*, Yamine Ait-Ameur and Shengchao Qin (Eds.). Springer International Publishing, Cham, 493–497.

[20] Hisham Shehata Galal, Yousef Bassyouni Mahdy, and Mohammed Ali Atiea. 2016. Behavior-based features model for malware detection. *Journal of Computer Virology and Hacking Techniques* 12 (2016), 59–67.

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

[22] Nikolai Hampton and Patryk Szewczyk. 2015. A survey and method for analysing soho router firmware currency. (2015).

[23] Philokypros Ioulianou, Vasileios Vasilakis, Ioannis Moscholios, and Michael Logothetis. 2018. A signature-based intrusion detection system for the internet of things. *Information and Communication Technology Form* (2018).

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

[25] BooJoong Kang, Suleiman Y Yerima, Kieran McLaughlin, and Sakir Sezer. 2016. N-opcode analysis for android malware classification and categorization. In *2016 International conference on cyber security and protection of digital services (cyber security)*. IEEE, 1–7.

[26] Feng Kang, Rong Jin, and Rahul Sukthankar. 2006. Correlated label propagation with application to multi-label learning. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, Vol. 2. IEEE, 1719–1726.

[27] Chuangfeng Li, Guangming Shen, and Wei Sun. 2021. Cross-architecture Intemet-of-Things malware detection based on graph neural network. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–7.

[28] Sen Li, Mengmeng Ge, Ruitao Feng, Xiaohong Li, and Kwok Yan Lam. 2023. Automatic Detection and Analysis towards Malicious Behavior in IoT Malware. In *2023 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 1332–1341.

[29] MaGraMal. 2024. *Appendix*. https://github.com/MaGraMal2024/MaGraMal/blob/main/Appendix.pdf

[30] MaGraMal. 2024. *Dataset*. https://github.com/MaGraMal2024/MaGraMal/tree/main/Dataset

[31] Meka. 2016. *Meka*. https://github.com/Waikato/meka/tree/f0cc96133399afa4c80d2b8a6342913fe9057fb0

[32] MITRE. 2013. ATT&CK. https://attack.mitre.org

[33] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005* (2017).

[34] Huy-Trung Nguyen, Quoc-Dung Ngo, and Van-Hoang Le. 2020. A novel graph-based approach for IoT botnet detection. *International Journal of Information Security* 19, 5 (2020), 567–577.

[35] Fan Ou and Jian Xu. 2022. S3Feature: A static sensitive subgraph-based feature for android malware detection. *Computers & Security* 112 (2022), 102513.

[36] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. 2016. IoTPOT: A novel honeypot for revealing current IoT threats. *Journal of Information Processing* 24, 3 (2016), 522–533.

[37] Qijing Qiao, Ruitao Feng, Sen Chen, Fei Zhang, and Xiaohong Li. 2022. Multi-label Classification for Android Malware Based on Active Learning. *IEEE Transactions on Dependable and Secure Computing* (2022).

[38] Radare2. 2006. *Radare2.* https://rada.re/r/

[39] Jesse Read, Peter Reutemann, Bernhard Pfahringer, and Geoff Holmes. 2016. Meka: a multi-label/multi-target extension to weka. *Journal of Machine Learning Research* 17, 21 (2016), 1–5.

[40] Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach. 2021. Adversarial machine learning attacks and defense methods in the cyber security domain. *ACM Computing Surveys (CSUR)* 54, 5 (2021), 1–36.

[41] Jiawei Su, Danilo Vargas Vasconcellos, Sanjiva Prasad, Daniele Sgandurra, Yaokai Feng, and Kouichi Sakurai. 2018. Lightweight classification of IoT malware based on image recognition. In *2018 IEEE 42Nd annual computer software and applications conference (COMPSAC)*, Vol. 2. IEEE, 664–669.

[42] Robin Taylor, David Baron, and Daniel Schmidt. 2015. The world in 2025-predictions for the next ten years. In *2015 10th International Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT)*. IEEE, 192–195.

[43] Sadegh Torabi, Mirabelle Dib, Elias Bou-Harb, Chadi Assi, and Mourad Debbabi. 2021. A strings-based similarity analysis approach for characterizing IoT malware and inferring their underlying relationships. *IEEE Networking Letters* 3, 3 (2021), 161–165.

[44] Grigorios Tsoumakas and Ioannis Katakis. 2007. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)* 3, 3 (2007), 1–13.

[45] Daniel Uhrıcek. 2020. Lisa–multiplatform linux sandbox for analyzing iot malware.

[46] Nighat Usman, Saeeda Usman, Fazlullah Khan, Mian Ahmad Jan, Ahthasham Sajid, Mamoun Alazab, and Paul Watters. 2021. Intelligent dynamic malware detection using machine learning in IP reputation for forensics data analytics. *Future Generation Computer Systems* 118 (2021), 124–141.

[47] J Vijayan. 2018. Satori Botnet Malware Now Can Infect Even More IoT Devices.

[48] VirusShare. 2012. *VirusShare.* https://virusshare.com/

[49] VirusTotal. 2004. *VirusTotal.* https://www.virustotal.com/

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

[51] Chia-Yi Wu, Tao Ban, Shin-Ming Cheng, Takeshi Takahashi, and Daisuke Inoue. 2023. IoT malware classification based on reinterpreted function-call graphs. *Computers & Security* 125 (2023), 103060.

[52] Fei Xiao, Yi Sun, Donggao Du, Xuelei Li, and Min Luo. 2020. A novel malware classification method based on crucial behavior. *Mathematical Problems in Engineering* 2020 (2020), 1–12.

[53] Yipin Zhang, Xiaolin Chang, Yuzhou Lin, Jelena Mišić, and Vojislav B Mišić. 2020. Exploring function call graph vectorization and file statistical features in malicious PE file classification. *IEEE Access* 8 (2020), 44652–44660.

[54] zynamics. 2011. *bindiff.* https://www.zynamics.com/bindiff.html

Received 2024-02-29; accepted 2024-04-01