IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 January 2023, accepted 9 February 2023, date of publication 22 February 2023, date of current version 28 February 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3247344

RESEARCH ARTICLE

Robust IoT Malware Detection and Classification Using Opcode Category Features on Machine Learning

HYUNJONG LEE[1], SOOIN KIM[2], DONGHEON BAEK[3], DONGHOON KIM[4], AND DOOSUNG HWANG[2]

[1]SANDS Laboratory, Seoul 06143, South Korea
[2]Department of Software Science, Dankook University, Yongin 16890, South Korea
[3]Dankook University, Cheonan 31119, South Korea
[4]Department of Computer Science, Arkansas State University, Jonesboro, AR 72401, USA

Corresponding authors: Doosung Hwang (dshwang@dankook.ac.kr) and Donghoon Kim (dhkim@astate.edu)

This work was supported by the Institute for Information and Communications Technology Promotion (IITP) Grant by the Korea Government [Ministry of Science and ICT (MIST)] (Trust-Based Cyber Security Platform for Smart Facility Environment) under Grant 2019-0-00197.

ABSTRACT Technology advancements have led to the use of millions of IoT devices. However, IoT devices are being exploited as an entry point due to security flaws by resource constraints. IoT malware is being discovered in a variety of types. The purpose of this study is to investigate whether IoT malware can be detected from benign and whether various malware family types can be classified. We propose fixed-length and low-dimensional features using opcode category information on ML models. The binary IoT dataset for this study is converted into opcode to create features. The opcodes are categorized into 6 or 11 according to their functionality. Features are created using a sequence of opcode categories and the entropy values of opcode categories. These features can be visualized by using a 2D image in order to observe patterns. We evaluate our proposed features on various ML models (5-NN, SVM, Decision Tree, and Random Forest) and MLP with various performance metrics, such as Accuracy, Precision, Recall, F1-score, MCC, AUC-ROC, and AUC-PR. The performance results for malware detection and classification have an accuracy over 98.0%. The experiments have demonstrated that the features we've proposed are effective and robust for identifying different types of IoT malware and benign.

INDEX TERMS IoT malware, machine learning, opcode category, sequence mining, visualization.

I. INTRODUCTION

IoT (Internet of Things) is the network of physical objects in which various objects use sensors to collect and share data via the Internet [1]. As IoT technology advances, the number of devices connected to the Internet increases. According to Statista [2], the number of IoT devices in 2022 is estimated to be 13.1 billion. In the meantime, IoT devices have become highly attractive targets for malware attacks due to their common vulnerabilities, such as easily guessed passwords, insufficient update security, and insecure network services. [3], [4], [5], [6], [7], [8]. As a result, hackers are using IoT devices as attack entry points.

It is challenging to develop a secure IoT device because of its inherent properties. IoT devices struggle to adhere to stringent security standards due to their limited resources and lack of a mechanism that automatically installs security patches [9]. There are many heterogeneous device architectures and network protocols [10]. The heterogeneity of IoT platforms makes it more difficult to detect IoT malware because various opcodes have equal functionality but different naming conventions [11]. As a result, the IoT industry falls behind the integrated design and implementation of security protocols. These weaknesses in IoT design increase the attack potential and lead to security leaks. These security

The associate editor coordinating the review of this manuscript and approving it for publication was Yassine Maleh.

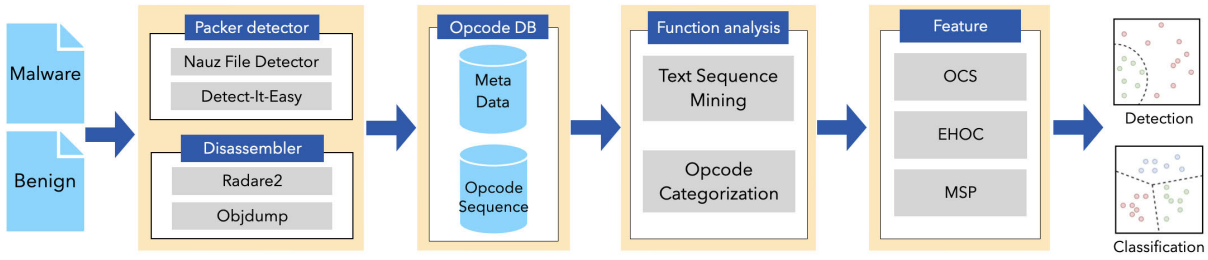


FIGURE 1. System workflow of the proposed approach.

flaws are exploited by hackers for illegal activities. Examples of major cybersecurity issues include host and network intrusions, malware attacks, botnets, rootkits, ransomware, and DDoS [12], [13], [14]. Therefore, a systematic approach to preventing these security exploits is required.

Building a secure defense system requires an in-depth knowledge of the operations and activities of malware. Key malware analysis should consider the factors that determine how malware works and how an attack affects a system. Such malware analysis is performed either statically or dynamically. Static analysis utilizes text-based structural relationship without executing it. The detection method is relationship analysis between low-level information, such as byte sequences [15], [16], *n*-gram [17], [18], [19], [20], [21], CFG (Control Flow Graph) [22], and system calls [23]. This approach is efficient and has a high detection rate with a low false positive rate. Static analysis traces all possible paths, so overall structural information within the malware is available. However, unknown malware designed to obfuscate code tends to go undetected. In dynamic analysis, malware is detected by analyzing the scan of infected files in a virtual machine. Although dynamic analysis is capable of detecting new and unknown malware, it is time-consuming and has a limitation of detecting only a few paths based on previously infected files [24].

Machine learning techniques have recently been extensively applied in malware detection due to their more robust and promising performance. Anti-malware vendors have increased the performance of malware analysis by employing machine learning technologies [36]. Furthermore, hardware improvements are also enhancing the performance of machine learning algorithms in malware analysis. A sufficient dataset and a fixed-length feature are required to utilize machine learning for IoT malware analysis. Each malware must be mapped to a fixed length vector that can encode its intrinsic structure or behavior. Nevertheless, it is challenging to provide the feature with a fixed-length because the size of each file is variable.

This paper deals with IoT malware detection and classification using opcode categories. The binary IoT malware is converted to a series of opcodes. Each opcode is converted into an opcode category classified according to its functionality. Features are created from the category by using frequent *n*-grams, maximal subpatterns, and entropy values. Most supervised learning algorithms are applicable for the designed feature with a fixed length. Figure 1 illustrates how the system works with malware detection and classification using our proposed features. The main contributions are described below:

- A novel feature design is proposed for analyzing malware in IoT devices by utilizing opcode category information. This approach simplifies malware features while maintaining their comparability for IoT malware analysis. The features derived from opcode categories can be applied to various types of IoT device architectures.
- The features obtained from the mapping process are represented with a fixed length, regardless of the original size of the malware files in IoT devices. These features capture the inherent characteristics of the malware.
- The proposed features can be represented as 2D images, which allows for the identification of common patterns and variations among different families of IoT malware. The visualization of the features also allows for easy interpretation of the prediction results of the models.
- The experimental results show that the proposed features have exceptional generalization abilities for detecting and classifying IoT malware, as evidenced by various performance metrics.

This paper is organized as follows. Section II discusses related work on IoT malware detection and classification. Section III analyzes various characteristics in the IoT malware dataset collected for these experiments. Section IV addresses IoT features and extraction methods for malware detection and classification and visualizes features. Section V describes the experimental procedures and evaluates each feature and machine learning models used for identifying IoT malware, and it compares the results with other studies. Finally, Section VI concludes this work with a summary.

II. RELATED WORK

Static analysis and dynamic analysis are methods for extracting features from executable files. We introduce a feature extraction method through static analysis in this study. Static analysis is a study that extracts various information from executable files, uses it to learn models, and detects and classifies malicious code. There are various features for static analysis. Many researchers employed binary and opcode based on

TABLE 1. Comparison of the studied malware analysis.

Study	Class	Total	Malware	Benign	Feature	Model	Source
Tien [25]	15	8,251	6,251	2,000	executable	CNN	IoTPOT [26] CZ.NIC [27]
McLaughlin [17]	2	19,170	9,902	9,268	opcode	CNN	Android Malware Genome project [28] McAfee Labs
Darabin [18]	2	516	247	269	MSP	Random Forest	VirusTotal [29]
Jung [15]	9	10,868	10,868	-	opcode byte length	CNN	Malware Challenge [30]
Lyda [31]	4	100	-	100	entropy	-	Windows XP Service Pack2
Sorokin [32]	-	-	-	-	entropy	-	-
Gibert [33]	9	10,868	10,867	-	entropy	CNN	Malware Challenge [30]
Paik [34]	9	10,868	10,868	-	entropy	CNN	Malware Challenge [30]
Our study	6	72,785	70,193	2,592	entropy	Tree-based Ensemble, MLP	Malwares [35]

n-gram as a feature. Tien et al. [25] used opcodes as features by dividing them into 12 types according to their description files and different ISAs (Instruction Set Architecture) since the IoT malicious samples were designed for different ISAs. Moon et al. [37] designed opcode sequences by categorizing opcode functionality. McLaughlin et al. [17] used gram-based CNN structure features to detect Android malware via MLP (Multi-Layer Perceptron). Darabian et al. [18] used opcode category sequences as malware features by applying MFP (Maximal Frequent Patterns). Yuxin and Siyi [19] disassembled Windows Portable Executable (PE) files to make opcodes and used opcode *n*-gram features. Dovom et al. [20] used a sequence of opcodes and reduced the dimensionality by selecting a part of opcodes based on information gain. Kang et al. [21] used *n*-gram opcode features to identify and categorize Android malware. They used 10-gram opcodes and found about 37M unique opcodes in their dataset of 2,520 malware samples. Jung et al. [15] used byte sequence structures, converted them into a 4-gram length sequence, and inserted into a hash function to form a 256 × 256 size hash map. Su et al. [38] designed 2D gray-scale image features from malware in terms of file size and byte sequence.

Entropy information is also an important component in the development of a feature. Lyda and Hamrock [31] used bintropy employing a binary-file entropy to calculate the amount of statistical variation types in a data stream. Sorokin [32] extracted the structural entropy from the byte sequence of the executable, used wavelet analysis to find the segment in the structural entropy, and compared the similarities by calculating the edit distance between executable files. Paik et al. [34] extracted structural entropy by calculating the structural entropy for each block for each byte. CFG (Control Flow Graph) is also one of the popular methods used to create a feature. Nguyen et al. [22] deployed IoT botnet detection using a printable string information (PSI) graph as a key feature. Abusnaina et al. [39] analyzed the robustness of CFG-based features to detect IoT malware. The characteristics of malware are also discovered using a variety of system information, such as file information along with section headers, symbolic sections, and program headers from ELF files [40].

Many studies use dynamic analysis using system information. Jeon et al. [23] proposed a dynamic analysis for IoT malware detection by using the following features: memory, network, VFS (virtual file system), process, and system calls. Rey et al. [41] used network traffic packets of IoT devices affected by malware as features. Researchers investigate the relationship or correlation between IoT malware since IoT malware reuses common functions. As a result, it is able to reconstruct family lineage, and trace evolution [37], [42], [43]. These studies contribute to the creation of features and improve the effectiveness of malware detection. Table 1 shows the comparison of the studied malware analysis using major items, such as the number of files within each dataset, along with the sources, features, and models that were used.

III. IoT MALWARE DATASET

A. DATASET ANALYSIS

Malwares provided a malware dataset that operates in the 32-bit ARM CPU framework [35]. IoT malware families were classified according to Kaspersky classification criteria.[1] Table 2 shows our dataset for this study. The total data size is 24,611, which includes 2,592 benign and 22,019 malware with 5 different types.

TABLE 2. IoT malware and benign dataset.

Type	Family	No.	Ratio (%)
Benign	-	2,592	10.53
Malware	Dofloo	1,071	4.35
	Gafgyt	9,325	37.89
	Mirai	9,460	38.44
	Nyadrop	1,620	6.58
	Tsunami	543	2.21
Total	-	24,611	100.00

IoT devices utilize a lightweight operating system (OS) that can function with less power, memory, and resources due to hardware limitations [44]. The majority of IoT devices use various processors, such as ARM, MIPS, SPARC, etc [45]. An execution file using dynamic links may only be executed

[1] https://securelist.com/ddos-attacks-in-q2-2021/103424/

on a system in which a library required for the execution exists. Executable files that use static linking are more likely to be executed regardless of whether the necessary library is installed in the system since the library is built into the executable. The number of IoT malware compiled through static linking is higher than the number of IoT malware compiled through dynamic linking [42], [45]. These analytical results appear to be a tactic to increase portability in order to raise the infection rate of IoT devices. `Radare2`[2] was used to check the linking method and whether the symbols could be stripped. Table 3 shows linking methods and whether or not it is a ''stripped''. The majority of IoT malware was compiled using a static linking, rather than a dynamic linking. `Dofloo` only used static linking. Most of the families, except `Mirai`, had a high rate of ''unstripped''.

`Objdump`[3] and `Radare2` disassemble opcode sequences from IoT binary files. When `objdump` reads a file, identifies the machine instructions encoded in the file, and converts them into opcode. All opcodes for each file are used for malware detection. Tables 4 and 5 show the file size (in KB) and opcode sequences of the IoT malware dataset. The size of the collected malware is different, so the opcode sequence shows differences. These differences are also observed among malware samples belonging to the same family, which makes it challenging to identify distinctive features for machine learning-based analysis. `Dofloo` typically has a larger file size than other families. Thus, the length of the opcode sequence extracted from the file is generally longer than in other families. `Nyadrop` has the smallest file size and opcode length among the malware dataset. `Gafgyt` and `Tsunami` have a similar average size of files and average length of opcode sequences, but `Gafgyt` has smaller standard deviation values for file and opcode sequence length than `Tsunami`. The smallest file size in `Mirai` is 1.04 KB, and the smallest opcode sequence length is 164. The smallest file size in `Tsunami` is 12.42 KB and the smallest opcode sequence is 64, which is sometimes smaller than `Mirai`.

B. PACKING ANALYSIS

Packing is one of the most sophisticated obfuscation techniques by malware authors using UPX (Ultimate Packer for eXecutables).[4] It is required to determine whether the executable file is encrypted or obfuscated in order to extract an opcode sequence. This can be determined by calculating the average entropy of byte sequences in the section area of the binary. Encryption or obfuscation algorithms typically have a high average entropy because they mix part or all of the existing byte sequences to appear random. On the other hand, such executable files have structural properties within randomness because they need to be decrypted or non-obfuscated [31]. To disable the standard UPX unpacking tool, malware authors can modify the magic number and UPX strings and insert junk bytes. The corrupted file blocks static malware analysis and also delays manual reverse engineering [46], [47].

We use the `Detect-It-Easy` (DIE)[5] and `Nauz File Detector` (NFD)[6] tools to identify which packer was used. If we are unable to identify which packer was used, we mark it as ''Unknown''. The majority of IoT malware is packed using UPX packers, and `Dofloo` used UPX to pack all of the files as seen in Table 6. Figure 2 shows the header of the ELF file, which is packed with UPX. The `l_info` and `p_info` structures, which are required for the UPX loader, are at the end of the program header table. The UPX header of 36 bytes is located at the end of the file. The `l_info` structure has information for the loader. The `p_info` structure has `p_filesizet` with file size information and `p_blocksize` with block size information. The UPX header has a magic number of byte sequence 0×55505821 which is ''UPX!'' in ASCII, the UPX packer version, format, file size, compression method, and compression level.

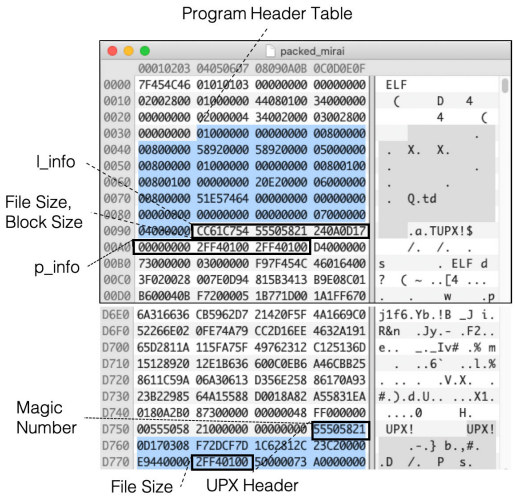


FIGURE 2. An example of valid ELF file using UPX.

We found a number of modified files that cannot be unpacked during our static analysis. Figure 3 shows a typical example of an invalid ELF file using UPX. The ''UPX!'' magic number of `l_magic` in the `l_info` structure is modified to 59525399 and `p_filesize` and `p_blocksize` of the `p_info` structure are modified to 00000000. However, unpacking with UPX can be performed by restoring the `p_filesize` and `p_blocksize` of the `p_info` structure of the UPX Header's `u_file_size` and changing the magic number to 55505821.

IV. RESEARCH APPROACH

This study will address the following malware detection and classification problems.

[2]https://rada.re/n/
[3]https://man7.org/linux/man-pages/man1/objdump.1.html
[4]https://upx.github.io/
[5]https://github.com/horsicq/Detect-It-Easy
[6]https://github.com/horsicq/Nauz-File-Detector

TABLE 3. Analysis of IoT malware linking methods.

Family	Dynamic Linking			Static Linking		
	Stripped	Unstripped	Total	Stripped	Unstripped	Total
Dofloo	0	0	0	1	1,070	1,071
Gafgyt	39	147	186	176	8,963	9,139
Mirai	1,261	0	1,261	4,346	3,853	8,199
Nyadrop	0	0	0	1,620	0	1,620
Tsunami	12	37	49	36	458	494
Total	1,312	184	1,496	4,559	14,344	18,903

TABLE 4. File size comparison of IoT malware families (KB).

Family	Min	Median	Max	Mean	Stdev
Dofloo	618.83	977.99	1026.33	941.44	97.16
Gafgyt	26.93	130.46	1897.20	134.62	51.16
Mirai	1.04	61	1865.10	79.06	80.62
Nyadrop	0.41	0.47	2.34	0.47	0.07
Tsunami	12.42	112.44	2720.05	137.62	141.04

TABLE 5. Opcode length comparison of IoT Malware families.

Family	Min	Median	Max	Mean	Stdev
Dofloo	102,396	137,776	147,813	133,189	9,490
Gafgyt	4,117	18,480	388,867	19,502	8,626
Mirai	164	14,207	381,916	15,273	15,297
Nyadrop	39	41	42	40	0.41
Tsunami	64	16,276	349,448	19,656	18,925

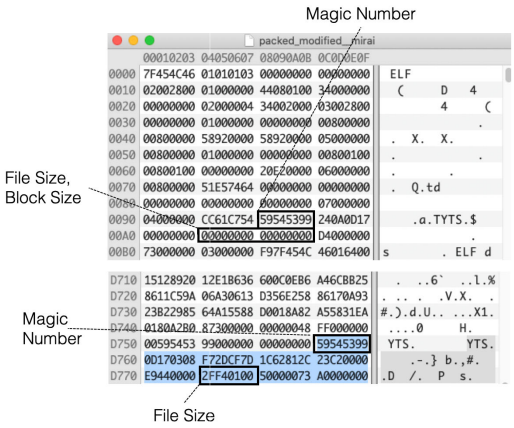


FIGURE 3. An example of invalid ELF file with manipulated UPX magic number.

- **Malware detection** is a binary classification. This study detects if a file is IoT malware or IoT benign. For this experiment, different IoT malware families are considered to be one label.
- **Malware classification** is a multi-classification. This study classifies 6 groups, including five malware families and one benign.

Let $\mathcal{X}$ be the dataset of malware and benign opcode sequences. Then, $\mathcal{X}$ consists of malware dataset $\mathcal{X}_{mal}$ and $\mathcal{X}_{ben}$.

$$\begin{aligned}\mathcal{X} &= \mathcal{X}_{mal} \cup \mathcal{X}_{ben} \\ &= \{(s_i, t) | i = 1, \ldots, N \text{ and } t \in \{\text{benign, malware}\}\},\end{aligned}$$

where $N = |\mathcal{X}_{mal}| + |\mathcal{X}_{ben}|$, $s_i =< op_1, op_2, \ldots, op_{i_n} >$ is the i^{th} opcode sequence and i_n is the length of s_i. Computing frequent subsequences from $\mathcal{X}_{mal}$, maximal subpatterns are discovered. $\mathrm{M}\mathcal{X}_{mal}$ is composed of maximal subpatterns that are decided from a text sequence mining tool. By applying MG-FSM [48], maximum subpatterns with the predefined minimum support rate min_{sup} or higher are extracted to construct $\mathrm{M}\mathcal{X}_{mal}$.

$$\mathrm{M}\mathcal{X}_{mal} = \{(msp_j, sup_j) | j = 1, \ldots, M \text{ and } sup_j \geq min_{sup}\}, \tag{1}$$

where msp_j is an opcode subpattern, M is the number of unique maximal subpatterns, and sup_j is the frequency rate in $\mathcal{X}_{mal}$. The opcode subpattern of msp_j is represented as $msp_j =< op_{j_1}, \ldots, op_{j_k} >$ and j_k is the length of msp_j.

Three types of malware features are derived from opcode category sequences: opcode category sequence (OCS), entropy histogram of opcode categories (EHOC), and maximal sequential pattern (MSP). Figure 4 illustrates the detailed process for extracting opcode category sequences.

TABLE 6. IoT malware packing analysis.

Family	Not packed	Packed		
		UPX	Unknown	Total
Dofloo	960	111	0	111
Gafgyt	9,071	252	2	254
Mirai	7,588	1,860	12	1,872
Nyadrop	1,620	0	0	0
Tsunami	475	50	18	68
Total	18,094	2,273	32	2,305

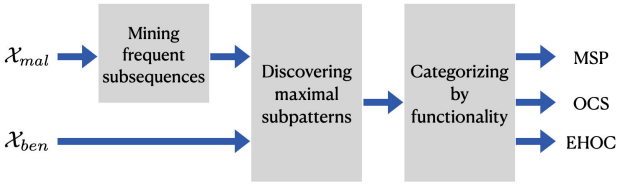


FIGURE 4. Feature extraction steps.

OCS: The opcode sequence s_i is converted into an opcode category sequence by utilizing the opcode category from MSP. These categories are distinguished by the opcode that make up all instances of $M_{\mathcal{X}_{mal}}$ in MSP. Table 7 and 8 illustrate the classification of 6 and 11 categories based on opcode function types. The opcode functions within these six categories are further divided into 11 categories. The opcode sequence s_i maps to the category sequence $s_i^{(C)}$ according to the described category. This mapping process simplifies the opcode sequence of malware.

The OCS feature of $s_i^{(C)}$ is constructed with a sliding window of size 256 and stride size 128. The frequency of n-grams appearing in each sliding window is accumulated and can be visualized as a 2D feature for $n = 2$. Both x-axis and y-axis of the 2D training feature are opcode categories. The category change of 2-gram reflects the structural change between category pairs through sliding window sequences. For $n = 2$ and K sliding windows, the OCS feature $F(s_i^{(C)})$ is computed as follows.

$$F(s_i^{(C)}) = [f_{jl}]_{\tau\times\tau} = \frac{1}{256K}\left[\sum_{k=1}^{K} \text{freq}_k(< c_j, c_l >)\right]_{\tau\times\tau}$$

$\text{freq}_k(< c_j, c_l >)$ is the frequency of 2-gram from c_j to c_l within sliding window k. When choosing 6 categories and $n = 2$, the dimension of OCS feature vector is 36, named $\text{OCS}_{n=2,c=6}$. In the case of 11 categories, it is expressed as $\text{OCS}_{n=2,c=11}$.

EHOC: EHOC features are constructed from opcode category sequences in a way similar to OCS. The entropy histogram map [49] is introduced to design feature vectors from opcode sequences. The entropy histogram E_i is computed from $s_i^{(C)}$ within a sliding window of size 256 and stride size 128. If $s_i^{(C)}$ has K windows, then $E_i =< w_1, \ldots, w_K >$ where $w_k =< e_1^{(k)}, \ldots, e_\tau^{(k)} >$ and $e_j^{(k)}$ is the entropy value of the j^{th} opcode category frequency within the k^{th} sliding window. The Shannon entropy is applied to compute $e_j^{(k)}$. Let h_j be the frequency rate of the j^{th} category.

$$e_j^{(k)} = -h_j \log_2 h_j - (1-h_j)\log_2(1-h_j) \tag{2}$$

Here, τ is 6 or 11, indicating the number of opcode categories (Table 8). EHOC feature $F(s_i^{(C)}) = [f_{\ell j}]_{L\times\tau}$ of $s_i^{(C)}$ becomes a 2D matrix. The column represents opcode category indexes, and the row indicates L discretized values. For a small constant δ, $F(s_i^{(C)})$ is calculated from all windows of E_j and $\ell = 1, 2, \ldots, L$.

$$F(s_i^{(C)}) = [f_{\ell j}]_{L\times\tau} \tag{3}$$

$$= \frac{1}{K}\left[\sum_{k=1}^{K} 1[\text{loc}(e_j^{(k)}) = (\ell, j)]e_j^{(k)}\right]_{L\times\tau} \tag{4}$$

If $(\ell-1)\delta < e_j^{(k)} \leq \ell\delta$, $\text{loc}(e_j^{(k)})$ returns (ℓ, j). $1[x]$ is 1 if x is true or 0 otherwise. The dimension of EHOC depends on the number of a discretized level L and opcode categories, and if L is 11 opcode categories, the number of EHOC feature dimensions is 121, which can be expressed as $\text{EHOC}_{l=11,c=11}$.

MSP: This feature is extended from Darabian et al. [18]. Every maximal subpattern is further represented by opcode function categories. All maximal subpatterns show a transition between one or two function categories. When only considering changes in two or fewer categories, the function category change from category i to j is defined as c_{i-j}. Assume there are τ function categories and C is a set of function types: $C = \{c_{i-j} | i, j = 1, \ldots, \tau\}$. The category c_{i-j} is a set of specific operations such as arithmetic operations, branch operations, function calls, etc. For $i = j$, c_{i-j} maximal subpattern is composed of operations by a single opcode category. Otherwise, c_{i-j} $(i \neq j)$ becomes a categorical transition from c_{i-i} to c_{j-j} in a maximal subpattern. Every msp_j is substituted with function categories: (c_{j-k}, sup_{j-k}). An opcode sequence is translated into a sequence of function categories. The opcode sequence of s_i turns into its category sequence $s_i^{(C)} =< c_{j_1-k_1}, \ldots, c_{j_\tau-k_\tau} >$. The feature vector of $s_i^{(C)}$ is $F(s_i^{(C)})$.

$$F(s_i^{(C)}) = [f_{j-k}]_{j,k=1,\ldots,\tau} = \frac{1}{|\mathcal{X}_{mal}|}\left[\text{freq}(c_{j-k}) \times sup_{j-k}\right]_{j,k=1,\ldots,\tau}$$

$\text{freq}(c_{j-k})$ is the number of occurring c_{j-k} in $s_i^{(C)}$. The number of $F(s_i^{(C)})$ is τ^2 which is much smaller in malware features such as byte sequence, opcode sequence, image feature, etc.

A maximal subpattern is extracted with a minimum support of 50% and $\tau = 6$. In the case of a 1-step transition MSP, there are a total of 36. Table 9 shows MSP and support rate explored by MG-FSM, where sup_{MSP} is the total number of MSP occurrences in the dataset, normalized to a scale from 0 to 1.

The proposed features can be visualized to identify key patterns when analyzing malware. Figure 5 shows the visualization of $\text{OCS}_{n=2,c=11}$ and $\text{EHOC}_{l=11,c=11}$ features. Each of the figures is an 11 × 11 2D image where the value of each feature vector was scaled between 0 and 1. Benign and malware are observed to be distinct. In addition, similar patterns are observed commonly throughout the malware family as a whole. The patterns on `Gafgyt` and `Tsunami` are comparable because they share a significant amount of code [37], [42].

V. EXPERIMENT

A. EXPERIMENTAL SETTING

Table 11 shows a detailed list of the IoT malware dataset for this experiment. For the malware detection experiment, all the data is used. On the other hand, for the malware classification experiment, we randomly select 3,000 datasets from each `Mirai` and `Gafgyt` randomly due to the high ratio of `Mirai` and `Gafgyt` for imbalanced data. As such,

TABLE 7. Opcode categories based on functionality.

High level	Low level	6 category	11 category
Branch instruction	-	c_3	c_1
Data-processing instructions	Standard data-processing instructions	c_1	c_2
	Shift instructions	c_6	c_3
	Multiplication instructions	c_2	c_4
	Packing and unpacking instructions	c_6	c_5
	Miscellaneous data-processing instructions	c_6	c_6
Status register access instructions	-	c_5	c_7
Load and store instruction	Load and store (single) instructions	c_4	c_8
	Load multiple and store multiple instructions		c_9
	Load and store coprocessor instructions		c_{10}
Other instructions	-	c_6	c_{11}

TABLE 8. Opcode list in functional categories.

Opcode	Category	Opcode
AND, EOR, SUB, RSB, ADD, ADC, SBC, RSC, TST, TEQ, CMP, CMN, ORR, MOV, BIC, MVN, CDP	c_1	B, BL, BLX, BX
MUL, MULL, MLA, MLAL	c_2	AND, EOR, SUB, RSB, ADD, ADC, SBC, RSC, TST, TEQ, CMP, CMN, ORR, MOV, MOVT, BIC, MVN, CDP
B, BL, BX	c_3	ASR, LSL, LSR, ROR, RRX
LDM, STM, LDR, STR, LDC, STC, MRC, MCR	c_4	MUL, MULL, MLA, MLAL
MRS, MSR	c_5	SXT, UXT
DMB, DSB, ISB, NOP, SEV, SVC, WFE, WFI, SWI, SWP, ADR, FLDM, YIELD	c_6	REV, REV16, REVSH
N/A	c_7	MRS, MSR, CPS
	c_8	LDR, STR
	c_9	LDM, STM, PUSH, POP
	c_{10}	LDC, STC, MRC, MCR
	c_{11}	DMB, DSB, ISB, NOP, SEV, SVC, WFE, WFI, UDF, SWI, SWP, ADR, FLDM, YIELD

TABLE 9. Examples of MSP and support rate.

MSP	Support	Type
ADD, SUB, RSB	0.901255	c_{1-1}
MUL, MUL	0.623923	c_{2-2}
B, B, B, B, B, B, B	0.514655	c_{3-3}
LDM, STM, LDR	0.575359	c_{4-4}
ADD, MUL	0.586113	c_{1-2}
ADD, SUB, ADD, B	0.639128	c_{1-3}
ADD, RSB, STR	0.546164	c_{1-4}
⋮	⋮	⋮

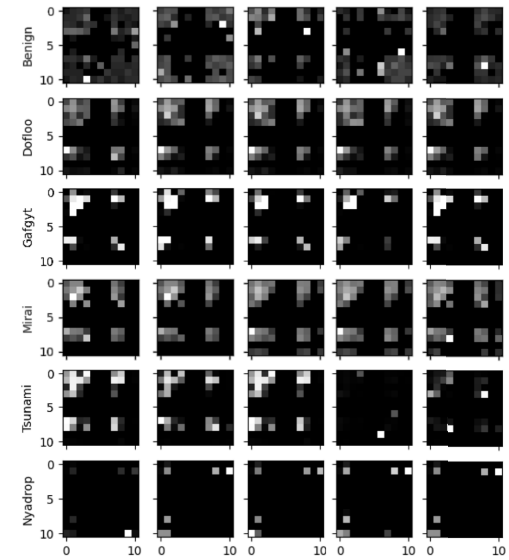
TABLE 10. Confusion matrix.

		Predicted class	
		Positive (P)	Negative (N)
Actual class	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

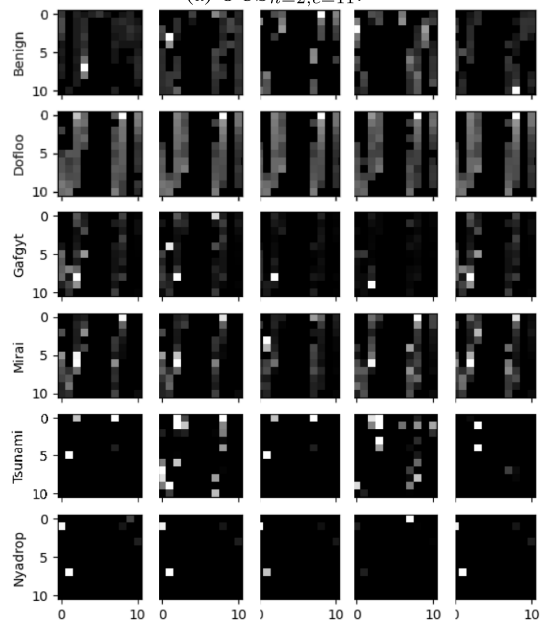
a total number of 11,826 malware classification experiments are conducted on 2,592 benign and 9,234 malware.

The following machine learning algorithms are utilized in the experiments: 5-NN, SVM, Decision Tree, Random Forest (RF), and MLP. To evaluate ML algorithms, each 5-fold CV runs 10 times, averaging the results. As performance metrics, accuracy (ACC), true positive rate (TPR), false positive rate (FPR), AUC-ROC, AUC-PR, F1-score (F1), and Matthews Correlation Coefficient (MCC) [50], precision (PRE), recall (REC) are chosen [47]. Table 10 presents a confusion matrix that indicates four different types of outcomes due to actual and predicted classifications.

MCC is chosen to evaluate imbalanced datasets and provide a balanced measure of performance since our dataset is imbalanced due to the nature of the data, which contains 2,592 benign and 9,234 malware. MCC is a more beneficial and honest metric for binary classification problems than accuracy or F1-score. MCC produces high metrics only when the predictions are accurate in all four categories of the confusion matrix. F1-score does not take into account how many true negatives are predicted, while MCC can be more accurate when evaluating imbalanced problems that require attention to negative data. Therefore, if a prediction model shows good performance in both F1-score and MCC, it can be considered a strong model for prediction. MCC is calculated using (5), and values range from -1 to $+1$, where $+1$ indicates a perfect prediction, -1 indicates a wrong prediction, and 0 indicates a random prediction.



(a) $\mathrm{OCS}_{n=2,c=11}$.



(b) $\mathrm{EHOC}_{l=11,c=11}$.

FIGURE 5. Feature visualization.

ACC is the proportion of correctly classified samples (both positive and negative) in a dataset. TPR, also known as sensitivity or recall, is the proportion of positive samples that are correctly classified as positive. FPR is the proportion of negative samples that are incorrectly classified as positive. AUC-ROC (Area Under the Receiver Operating Characteristic curve) is a metric for binary classification that represents the ability of a model to distinguish between positive and negative classes. It is calculated by plotting the true positive rate (y-axis) against the false positive rate (x-axis) at different classification thresholds. AUC-PR (Area Under the Precision-Recall curve) is a metric for binary classification that represents the trade-off between precision and recall for different classification thresholds. It is calculated by plotting the precision (y-axis) against the recall (x-axis). F1 is a measure of a model's accuracy that considers both the precision and recall of the model. It is calculated as the harmonic mean of precision and recall. PRE is the proportion of samples classified as positive that is actually positive. REC, also known as sensitivity or true positive rate, is the proportion of positive samples that are correctly classified as positive.

$$MCC = \frac{TN \times TP - FN \times FP}{\sqrt{(TP+FP)(TP+FN)}} \times \frac{1}{\sqrt{(TN+FP)(TN+FN)}} \tag{5}$$

Each experiment uses five features as follows.

- $\mathrm{OCS}_{n=2,c=6}$: OCS is 2-gram feature generated from 6 opcode category.
- $\mathrm{OCS}_{n=2,c=11}$: OCS is 2-gram feature generated from 11 opcode category.
- $\mathrm{EHOC}_{l=6,c=6}$: EHOC is a feature generated from 6 opcode category using 6 levels.
- $\mathrm{EHOC}_{l=11,c=11}$: EHOC is a feature generated from 11 opcode category using 11 levels.
- MSP: MSP feature was proposed by Darabian et al. [18]. This feature is used to compare with the four features listed above.

B. MALWARE DETECTION

The experimental results for malware detection are shown in Table 13. The parameters for the Support Vector Machine (SVM) are set to a value of c=1000 and use Radial Basis Function (RBF) kernels with a gamma value of "scale". The decision tree algorithm uses the Gini impurity as the split criterion, has a maximum depth of 10, and a minimum sample size of 2. The Random Forest (RF) model uses 100 decision trees with the same parameters as a single decision tree. The MLP architecture is $64 \times 12 \times 2$ for the first hidden layer and is optimized using the Adam algorithm with a learning rate of 0.001. The activation function for the hidden layers is ReLU and for the output layer is softmax.

In general, ML models perform well. The performance of RF stands out among them. Since the dataset is imbalanced, ACC and MCC are mainly analyzed. MSP has 99.5% ACC and 98.5% MCC in RF. $\mathrm{OCS}_{n=2,c=6}$ has 99.5% ACC and 98.6% MCC in RF. $\mathrm{OCS}_{n=2,c=11}$ has 99.8% ACC and 99.3% MCC in RF, which is the highest performance. $\mathrm{EHOC}_{l=6,c=6}$ has 99.0% ACC and 97.1% MCC in RF. $\mathrm{EHOC}_{l=11,c=11}$ has 99.6% ACC and 98.9% MCC in RF. 11 categories perform somewhat better than 6 categories in each feature. In addition, $\mathrm{OCS}_{n=2,c=11}$ and $\mathrm{EHOC}_{l=11,c=11}$ perform slightly better than MSP.

Figure 6 shows ROC analysis for the results of binary classification, which plots TPR against FPR. TPR is the proportion of observations that were correctly predicted to be positive out of all positive observations, which is calculated using (6). Similarly, FPR is the proportion of observations

TABLE 11. Experimental dataset.

Type	Family	Total		Detection		Classification	
		No.	Ratio (%)	No.	Ratio (%)	No.	Ratio (%)
Benign	-	2,592	10.53	2,592	10.53	2,592	21.92
Malware	Dofloo	1,071	4.35	22,019	89.47	1,071	9.06
	Gafgyt	9,325	37.89			3,000	25.37
	Mirai	9,460	38.44			3,000	25.37
	Nyadrop	1,620	6.58			1,620	13.70
	Tsunami	543	2.21			543	4.59
Total		24,611	100.00	24,611	100.00	11,826	100.00

TABLE 12. Dimension for feature vectors.

Feature	Dimension
$OCS_{n=2,c=6}$	36
$OCS_{n=2,c=11}$	121
$EHOC_{l=6,c=6}$	36
$EHOC_{l=11,c=11}$	121
MSP	36

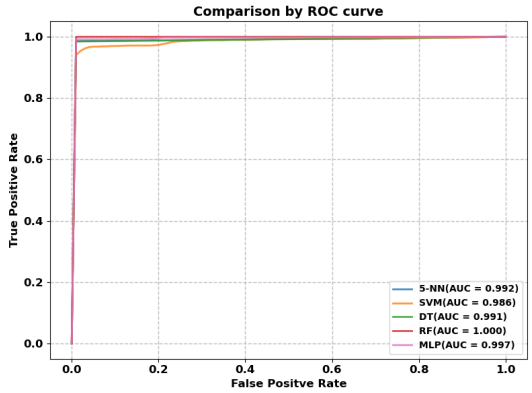


FIGURE 6. ROC analysis with $OCS_{n=2,c=11}$.

that are incorrectly predicted to be positive out of all negative observations, which is calculated using (7). In RF, the ROC value is 100.0% which indicates the best performance in binary classification.

$$TPR = \frac{TP}{TP + FN} \tag{6}$$

$$FPR = \frac{FP}{TN + FP} \tag{7}$$

C. MALWARE CLASSIFICATION

The experimental results for malware classification, which is a multi-classification with 6 different types, including 5 malware families and 1 benign, are shown in Table 13. In the classification experiment, the parameters for the chosen algorithm are similar across models. However, the classification task involves predicting 6 classes. The architecture of the Multi-Layer Perceptron (MLP) for this task is $64 \times 12 \times 6$ for the first hidden layer. The performance of multi-classification is slightly less than that of binary classification by a tiny margin of roughly 1%. Similar to malware detection, the performance of RF is the best among the ML models. The experimental results are analyzed, focusing on ACC and MCC. MSP has 98.5% ACC and 97.9% MCC in RF. $OCS_{n=2,c=6}$ has 98.1% ACC and 97.4% MCC in RF. $OCS_{n=2,c=11}$ has 98.6% ACC and 98.1% MCC in RF, which is the highest performance. $EHOC_{l=6,c=6}$ has 97.6% ACC and 96.7% MCC in RF. $EHOC_{l=11,c=11}$ has 98.1% ACC and 97.4% MCC in RF. As with experiments in malware detection, 11 categories perform slightly better than 6 categories in each feature. MSP performs similarly to other models.

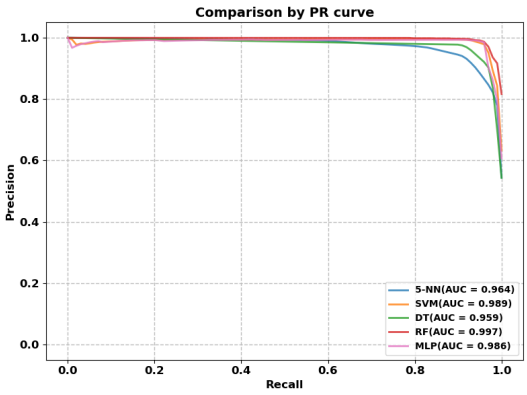


FIGURE 7. PR-AUC analysis with $OCS_{n=2,c=11}$.

Figure 7 shows PR-AUC analysis, which is a reliable indicator of performance for imbalanced classification problems. PR-AUC combines precision and recall in a single visualization, which is calculated using (8) and (9). The higher the curve on the y-axis, the better the performance. The scores in most of the models are higher than 99%, which indicates that the models employing our proposed features perform quite well.

$$PRE = \frac{TP}{TP + FP} \tag{8}$$

$$REC = \frac{TP}{TP + FN} \tag{9}$$

D. COMPARISON WITH OTHER STUDIES

The results of the proposed method were compared with similar studies, such as opcode feature [25], 2D image of

TABLE 13. Experimental results for malware detection with percentage (%).

Feature	Model	ACC	TPR	FPR	AUC-ROC	F1	MCC
$OCS_{n=2,c=6}$	5-NN	98.5	99.8	6.1	99.0	97.7	95.5
	SVM	93.8	99.7	26.0	97.4	90.4	82.1
	DT	99.0	99.5	2.9	98.3	98.5	97.0
	RF	99.5	99.7	1.2	100.0	99.3	98.6
	MLP	99.0	99.7	3.7	99.6	98.5	97.0
$OCS_{n=2,c=11}$	5-NN	98.9	99.8	4.4	99.2	98.4	96.8
	SVM	95.8	99.9	18.3	98.6	93.5	87.8
	DT	99.5	99.7	1.5	99.1	99.2	98.4
	RF	99.8	99.8	0.4	100.0	99.7	99.3
	MLP	99.5	99.8	1.5	99.7	99.3	98.7
$EHOC_{l=6,c=6}$	5-NN	97.6	99.7	9.6	98.1	96.4	93.0
	SVM	96.3	99.0	12.7	97.2	94.6	89.3
	DT	98.1	99.1	5.0	97.0	97.3	94.7
	RF	99.0	99.4	2.5	99.9	98.5	97.1
	MLP	98.6	99.3	4.0	99.5	97.9	95.8
$EHOC_{l=11,c=11}$	5-NN	98.2	99.6	6.3	98.7	97.4	94.9
	SVM	98.2	99.3	5.8	98.8	97.4	94.8
	DT	99.2	99.6	2.1	98.7	98.9	97.7
	RF	99.6	99.7	0.7	100.0	99.5	98.9
	MLP	99.3	99.7	1.9	99.7	99.0	98.1
MSP	5-NN	98.6	99.6	4.9	99.3	98.0	96.0
	SVM	98.8	99.3	2.6	99.3	98.3	96.6
	DT	99.0	99.4	2.5	98.4	98.6	97.1
	RF	99.5	99.6	1.0	100.0	99.2	98.5
	MLP	99.1	99.4	2.0	99.7	98.7	97.4

TABLE 14. Experimental results for malware classification with percentage (%).

Feature	Model	ACC	PRE	REC	AUC-PR	F1	MCC
$OCS_{n=2,c=6}$	5-NN	97.0	96.5	94.8	96.0	95.5	95.9
	SVM	97.6	97.2	97.1	98.6	97.2	96.8
	DT	97.1	95.9	96.2	93.1	96.1	96.0
	RF	98.1	98.3	96.7	99.3	97.5	97.4
	MLP	98.1	97.9	97.8	98.4	97.9	97.4
$OCS_{n=2,c=11}$	5-NN	97.0	96.6	93.7	96.4	94.9	95.9
	SVM	98.4	98.2	98.2	98.9	98.1	97.8
	DT	98.1	97.6	97.8	95.9	97.7	97.4
	RF	98.6	98.6	98.4	99.7	98.5	98.1
	MLP	98.6	98.5	98.5	98.6	98.5	98.1
$EHOC_{l=6,c=6}$	5-NN	95.3	93.8	90.5	93.1	91.8	93.5
	SVM	97.2	96.0	95.2	97.1	95.6	96.2
	DT	96.1	93.9	93.7	89.5	93.8	94.7
	RF	97.6	97.6	95.2	98.7	96.2	96.7
	MLP	97.6	96.8	96.0	97.8	96.4	96.7
$EHOC_{l=11,c=11}$	5-NN	95.5	94.1	90.3	92.8	91.8	93.8
	SVM	97.8	97.4	96.8	98.8	97.1	97.0
	DT	96.7	94.6	94.9	90.9	94.7	95.5
	RF	98.1	98.1	96.5	99.2	97.2	97.4
	MLP	97.8	97.1	96.7	98.5	96.9	96.9
MSP	5-NN	97.8	97.9	97.2	98.3	97.5	97.0
	SVM	98.0	97.9	97.6	98.7	97.8	97.3
	DT	97.7	97.2	97.3	95.2	97.3	96.9
	RF	98.5	98.3	98.4	99.7	98.3	97.9
	MLP	98.3	98.1	98.3	98.8	98.2	97.7

executable file [51], behavior-based feature [23], CFG feature [52], etc. We selected to use a MLP model as the comparison models employed a Convolutional Neural Network (CNN) as the deep learning algorithm. The use of MLP allows for an efficient analysis of the low-dimensional features created in this work, without the need for a CNN. Table 15 shows comparison factors including model, feature type, dataset composition, number of malware families, accuracy,

TABLE 15. Analysis of comparison with other studies.

Study		Tien [25]	Jeon [23]	Asam [51]	Alasmary [52]	Our study
Model		CNN	CNN	CNN	CNN	MLP
Analysis		Detection	Detection	Detection	Detection/Classification	Detection/Classification
Feature		Opcode	Behavior	Executable	CFG	Opcode category
Dataset	Benign	2,157	401	2,486	2,999	2,592
	Malware	6,251	1,000	14,733	2,962	8,940
No. of families		-	-	-	4	6
Accuracy		99.0	99.28	97.93	99.66/99.32	99.7/98.1
F1-score		97.0	99.94	93.94	99.99/99.63	99.3/97.9

and F1-score. Alasmary et al. [52] and our proposed method include both malware detection and family classification analysis, and other studies have reported detection analysis only. Thus, two numbers in Accuracy and F1-score are binary classifications for malware detection and multi-classification. Jeon et al. [23] configured features by extracting information such as files, networks, and system calls through debugging. Asam et al. [51] performed CNN model analysis by extracting 2D image features of executable files. Alasmary et al. [52] suggested CFG features from opcodes and performed various model analysis, but the CNN model had excellent performance.

The proposed method represents IoT malware with fixed-length and low-dimensional features, and most standard supervised learning can be applied. Our experiments performed malware classification using the largest number of families and sufficient amounts of data. The experimental results demonstrated that our fixed-length features provide inherent structural characteristics for variable lengths of IoT malware files. In addition, the proposed features using opcode categories are expressed by fewer elements compared to the opcode sequence, resulting in the advantage of reduced training time due to the low dimension.

This comparison has a limitation in that the datasets used are not the same. Our model has similar performance to other models in terms of accuracy and F1-score. Although CNN models are known for being computationally intensive, they usually have good performance. Our model's advantage is less computationally demand because of low-dimensional features (36 and 121 features as shown in Table 12). However, we do not compare the time complexity of our model to other models as the datasets used in the comparison are different and we lack information about the time complexity of other models.

VI. CONCLUSION

This study presented an effective and robust approach for IoT malware detection and classification. The proposed features were created using opcode categories based on opcode functionality. The features were simplistic, fixed-length, and low-dimensional despite the variable length of IoT malware. We analyzed IoT malware dataset using various methods, such as file information and file structure when packing. We addressed in detail how to create features using opcode categories that can represent the characteristics of IoT malware. The features were visualized, so common patterns and differences were observed. We thoroughly evaluated the effectiveness and robustness of our proposed features using a large dataset of IoT malware, various performance metrics, and ML models, including tree-based ensemble models and MLP.

Our research has made a significant contribution to the field of malware detection with the development of a robust method for creating fixed and low-dimensional features from malware of varying file sizes. This technique is used to represent the characteristics of malware in a compact, fixed-length format. It allows for accurate representation of malware characteristics for efficient detection and analysis. This is a crucial step towards better protecting individuals and organizations from cyber threats and is expected to have a major impact on the field of malware analysis research. Our method is reliable and has the potential to greatly advance the current state of the art in this area.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[2] Statista. *Number of Internet of Things (IoT) Connected Devices Worldwide From 2019 to 2021, With Forecasts From 2022 to 2030*. Accessed: Sep. 20, 2022. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/#main-content

[3] S. Singh and N. Singh, "Internet of Things (IoT): Security challenges, business opportunities & reference architecture for e-commerce," in *Proc. Int. Conf. Green Comput. Internet Things (ICGCIoT)*, Oct. 2015, pp. 1577–1581.

[4] X. Jiang, M. Lora, and S. Chattopadhyay, "An experimental analysis of security vulnerabilities in industrial IoT devices," *ACM Trans. Internet Technol.*, vol. 20, no. 2, pp. 1–24, May 2020.

[5] E. Bertino and N. Islam, "Botnets and Internet of Things security," *Computer*, vol. 50, no. 2, pp. 76–79, Feb. 2017.

[6] A. S. Rajawat, R. Rawat, K. Barhanpurkar, R. N. Shaw, and A. Ghosh, "Blockchain-based model for expanding IoT device data security," in *Advances in Applications of Data-Driven Computing*. Singapore: Springer, 2021, pp. 61–71.

[7] Q. Abu Al-Haija and M. Al-Dala'ien, "ELBA-IoT: An ensemble learning model for botnet attack detection in IoT networks," *J. Sensor Actuator Netw.*, vol. 11, no. 1, p. 18, Mar. 2022.

[8] K. Albulayhi, Q. Abu Al-Haija, S. A. Alsuhibany, A. A. Jillepalli, M. Ashrafuzzaman, and F. T. Sheldon, "IoT intrusion detection using machine learning with a novel high performing feature selection method," *Appl. Sci.*, vol. 12, no. 10, p. 5015, May 2022.

[9] L. Wüstrich, M.-O. Pahl, and S. Liebald, "Towards an extensible IoT security taxonomy," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2020, pp. 1–6.

[10] M. M. Hossain, M. Fotouhi, and R. Hasan, "Towards an analysis of security issues, challenges, and open problems in the Internet of Things," in *Proc. IEEE World Congr. Services*, Jun. 2015, pp. 21–28.
[11] Y.-T. Lee, T. Ban, T.-L. Wan, S.-M. Cheng, R. Isawa, T. Takahashi, and D. Inoue, "Cross platform IoT-malware family classification based on printable strings," in *Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Dec. 2020, pp. 775–784.
[12] D. Kim and M. A. Vouk, "A survey of common security vulnerabilities and corresponding countermeasures for SaaS," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2014, pp. 59–63.
[13] D. Kim, H. E. Schaffer, and M. A. Vouk, "About PaaS security," *Int. J. Cloud Comput.*, vol. 6, no. 4, pp. 325–341, 2017.
[14] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.
[15] B. Jung, T. Kim, and E. G. Im, "Malware classification using byte sequence information," in *Proc. Conf. Res. Adapt. Convergent Syst.*, Oct. 2018, pp. 143–148.
[16] Z. Moti, S. Hashemi, H. Karimipour, A. Dehghantanha, A. N. Jahromi, L. Abdi, and F. Alavi, "Generative adversarial network to detect unseen Internet of Things malware," *Ad Hoc Netw.*, vol. 122, Nov. 2021, Art. no. 102591.
[17] N. McLaughlin, J. M. Del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé, and G. J. Ahn, "Deep Android malware detection," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, 2017, pp. 301–308.
[18] H. Darabian, A. Dehghantanha, S. Hashemi, S. Homayoun, and K. R. Choo, "An opcode-based technique for polymorphic Internet of Things malware detection," *Concurrency Comput., Pract. Exp.*, vol. 32, no. 6, p. e5173, Mar. 2020.
[19] D. Yuxin and Z. Siyi, "Malware detection based on deep learning algorithm," *Neural Comput. Appl.*, vol. 31, no. 2, pp. 461–472, Feb. 2019.
[20] E. M. Dovom, A. Azmoodeh, A. Dehghantanha, D. E. Newton, R. M. Parizi, and H. Karimipour, "Fuzzy pattern tree for edge malware detection and categorization in IoT," *J. Syst. Archit.*, vol. 97, pp. 1–7, Aug. 2019.
[21] B. Kang, S. Y. Yerima, S. Sezer, and K. McLaughlin, "N-gram opcode analysis for Android malware detection," *Int. J. Cyber Situational Awareness*, vol. 1, no. 1, pp. 231–255, Dec. 2016.
[22] H.-T. Nguyen, Q.-D. Ngo, and V.-H. Le, "IoT botnet detection approach based on PSI graph and DGCNN classifier," in *Proc. IEEE Int. Conf. Inf. Commun. Signal Process. (ICICSP)*, Sep. 2018, pp. 118–122.
[23] J. Jeon, J. H. Park, and Y.-S. Jeong, "Dynamic analysis for IoT malware detection with convolution neural network model," *IEEE Access*, vol. 8, pp. 96899–96911, 2020.
[24] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, "Malware dynamic analysis evasion techniques: A survey," *ACM Comput. Surveys*, vol. 52, no. 6, pp. 1–28, Nov. 2020.
[25] C.-W. Tien, S.-W. Chen, T. Ban, and S.-Y. Kuo, "Machine learning framework to analyze IoT malware using ELF and opcode features," *Digit. Threats, Res. Pract.*, vol. 1, no. 1, pp. 1–19, Mar. 2020.
[26] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "IoTPOT: Analysing the rise of IoT compromises," in *Proc. 9th USENIX Workshop Offensive Technol. (WOOT)*, 2015, pp. 1–9.
[27] *Haas*. Accessed: Sep. 20, 2021. [Online]. Available: https://haas.nic.cz/
[28] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 95–109.
[29] *Virustotal*. Accessed: Sep. 20, 2021. [Online]. Available: https://www.virustotal.com/
[30] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," 2018, *arXiv:1802.10135*.
[31] R. Lyda and J. Hamrock, "Using entropy analysis to find encrypted and packed malware," *IEEE Secur. Privacy*, vol. 5, no. 2, pp. 40–45, Mar./Apr. 2007.
[32] I. Sorokin, "Comparing files using structural entropy," *J. Comput. Virol.*, vol. 7, no. 4, pp. 259–265, Nov. 2011.
[33] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Classification of malware by using structural entropy on convolutional neural networks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 1–6.
[34] J. Paik, R. Jin, and E. Cho, "Malware classification using a byte-granularity feature based on structural entropy," *Comput. Intell.*, vol. 38, no. 4, pp. 1536–1558, Aug. 2022.
[35] S Lab. *Malwares*. Accessed: Sep. 20, 2021. [Online]. Available: http://www.malwares.com/
[36] H. S. Anderson, A. Kharkar, B. Filar, and P. Roth, "Evading machine learning malware detection," *Black Hat*, vol. 2017, pp. 1–6, Jul. 2017.
[37] S. Moon, Y. Kim, H. Lee, D. Kim, and D. Hwang, "Evolved IoT malware detection using opcode category sequence through machine learning," in *Proc. Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2022, pp. 1–7.
[38] J. Su, V. D. Vasconcellos, S. Prasad, S. Daniele, Y. Feng, and K. Sakurai, "Lightweight classification of IoT malware based on image recognition," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2018, pp. 664–669.
[39] A. Abusnaina, A. Khormali, H. Alasmary, J. Park, A. Anwar, and A. Mohaisen, "Adversarial learning attacks on graph-based IoT malware detection systems," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 1296–1305.
[40] F. Shahzad and M. Farooq, "ELF-miner: Using structural knowledge and data mining methods to detect new (Linux) malicious executables," *Knowl. Inf. Syst.*, vol. 30, no. 3, pp. 589–612, Mar. 2012.
[41] V. Rey, P. M. S. Sánchez, A. H. Celdrán, and G. Bovet, "Federated learning for malware detection in IoT devices," *Comput. Netw.*, vol. 204, Feb. 2022, Art. no. 108693.
[42] E. Cozzi, P.-A. Vervier, M. Dell'Amico, Y. Shen, L. Bilge, and D. Balzarotti, "The tangled genealogy of IoT malware," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2020, pp. 1–16.
[43] M. Dib, S. Torabi, E. Bou-Harb, N. Bouguila, and C. Assi, "EVOLIoT: A self-supervised contrastive learning framework for detecting and characterizing evolving IoT malware variants," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, May 2022, pp. 452–466.
[44] F. Javed, M. K. Afzal, M. Sharif, and B.-S. Kim, "Internet of Things (IoT) operating systems support, networking technologies, applications, and challenges: A comparative review," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2062–2100, 3rd Quart., 2018.
[45] O. Alrawi, C. Lever, K. Valakuzhy, Ryan Court, K. Z. Snow, F. Monrose, and M. Antonakakis, "The circle life: A large-scale study of the IoT malware lifecycle," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.)*, 2021, pp. 3505–3522.
[46] E. Cozzi, *Binary Analysis for Linux and IoT Malware*. Paris, France: Sorbonne Université, 2020.
[47] J. D. Kelleher, B. M. Namee, and A. DArcy, *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*, 2nd ed. Cambridge, MA, USA: MIT Press, 2020.
[48] I. Miliaraki, K. Berberich, R. Gemulla, and S. Zoupanos, "Mind the gap: Large-scale frequent sequence mining," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 2013, pp. 797–808.
[49] S. Euh, H. Lee, D. Kim, and D. Hwang, "Comparative analysis of low-dimensional features and tree-based ensembles for malware detection systems," *IEEE Access*, vol. 8, pp. 76796–76808, 2020.
[50] S. Boughorbel, F. Jarray, and M. El-Anbari, "Optimal classifier for imbalanced data using Matthews correlation coefficient metric," *PLoS ONE*, vol. 12, no. 6, pp. 1–17, 2017.
[51] M. Asam, S. H. Khan, A. Akbar, S. Bibi, T. Jamal, A. Khan, U. Ghafoor, and M. R. Bhutta, "IoT malware detection architecture using a novel channel boosted and squeezed CNN," *Sci. Rep.*, vol. 12, no. 1, pp. 1–12, Sep. 2022.
[52] 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 Things J.*, vol. 6, no. 5, pp. 8977–8988, Oct. 2019.

HYUNJONG LEE received the M.S. degree from Dankook University, South Korea. He worked with the Security Technology Institute, KSIGN, as a Junior Researcher. He is currently a Research Staff Member with SANDS Laboratory, South Korea. His research interests include machine learning, representation learning, and distributed computing.

SOOIN KIM received the B.S. degree from Dankook University, South Korea, where he is currently pursuing the degree with the Department of Computer Science. His research interests include neural network architecture, data structure, and software engineering.

DONGHEON BAEK received the Ph.D. degree from Seoul National University, South Korea. He was a Public Health Doctor with the Korean Food and Drug Administration and worked on regulatory affairs of medical devices certification. He is currently a Professor with the Department of Oral Microbiology and Immunology, School of Dentistry, Dankook University, South Korea. His research interests include the T-cell immunity, pathogenicity of periodontal diseases, evaluation of public healthcare devices, bioinformatics and applications, and information systems.

DONGHOON KIM received the Master of Science degree from Auburn University, USA, and the Ph.D. degree from North Carolina State University, USA. During his Ph.D. degree, he worked with SAS Institute and IBM Cloud Computing Team as a Research Intern. Previously, he was a Software Engineer at Samsung Electronics, South Korea. He is currently an Associate Professor with the Department of Computer Science, Arkansas State University, USA. His research interests include computer security, high-performance computing, and software engineering. He serves as an Associate Editor for *JIPS* journal.

DOOSUNG HWANG received the Ph.D. degree from Wayne State University, USA. Previously, he was a Senior Researcher with Electronics and Telecommunications Research Institute (ETRI), South Korea, and worked on learning algorithm design and intelligent systems, such as expert systems, image recognition, time-series analysis, and parallel computing. He is currently a Professor with the Department of Software Science, Dankook University, South Korea.