IEEE Open Journal of the Computer Society

Received 23 July 2020; accepted 10 October 2020. Date of publication 26 October 2020;
date of current version 23 November 2020. The review of this paper was arranged by Associate Editor Dr. Song Guo.

Digital Object Identifier 10.1109/OJCS.2020.3033974

Efficient Detection and Classification of Internet-of-Things Malware Based on Byte Sequences from Executable Files

TZU-LING WAN [1], TAO BAN [3] (Member, IEEE), SHIN-MING CHENG [1,2] (Member, IEEE), YEN-TING LEE [1], BO SUN [4], RYOICHI ISAWA [3], TAKESHI TAKAHASHI [3] (Member, IEEE), AND DAISUKE INOUE [3] (Member, IEEE)

[1]Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei 106, Taiwan
[2]Research Center for Information Technology Innovation, Academia Sinica, Taipei 106, Taiwan
[3]National Institute of Information and Communications Technology, Koganei, Tokyo 184-8795, Japan
[4]Saitama Institute of Technology, Saitama 369-0293, Japan

CORRESPONDING AUTHOR: TAO BAN (e-mail: bantao@nict.go.jp)

This work was supported by the Ministry of Science and Technology, Taiwan, under Grants 109-2636-E-009-022 and 109-2218-E-011-007.

ABSTRACT Simple implementation and autonomous operation features make the Internet-of-Things (IoT) vulnerable to malware attacks. Static analysis of IoT malware executable files is a feasible approach to understanding the behavior of IoT malware for mitigation and prevention. However, current analytic approaches based on opcodes or call graphs typically do not work well with diversity in central processing unit (CPU) architectures and are often resource intensive. In this paper, we propose an efficient method for leveraging machine learning methods to detect and classify IoT malware programs. We show that reliable and efficient detection and classification can be achieved by exploring the essential discriminating information stored in the byte sequences at the entry points of executable programs. We demonstrate the performance of the proposed method using a large-scale dataset consisting of 111K benignware and 111K malware programs from seven CPU architectures. The proposed method achieves near optimal generalization performance for malware detection (99.96% accuracy) and for malware family classification (98.47% accuracy). Moreover, when CPU architecture information is considered in learning, the proposed method combined with support vector machine classifiers can yield even higher generalization performance using fewer bytes from the executable files. The findings in this paper are promising for implementing light-weight malware protection on IoT devices with limited resources.

INDEX TERMS Computer security, machine learning, binary code, malware analysis, static analysis.

I. INTRODUCTION

Recent proliferation of Internet of Things (IoT) devices brought revolutionary change to information processing and daily life. A survey from Gartner [1] predicted that the number of IoT-enabled devices will reach 24 billion by 2020. This number was, in fact, surpassed by 10% by the end of 2019 as reported in [2]. This exponential growth in popularity of IoT applications and devices has given rise to new security challenges. With heterogeneous central processing unit (CPU) architectures, constrained resources, limited interfaces, unsafe default configurations, and difficult-to-patch software implementations, IoT devices have been found to be vulnerable to many critical security breaches [3], [4]. According to a report of security researchers from F-Secure, there were 2.8 billion cyberattack packets observed at their global honeypot network in the second half of 2019 [5]. Moreover, the source-code release in the infamous Mirai attack triggered the spread of IoT attacks and the growth of malware families [6]–[8]. How to protect vulnerable IoT devices from being compromised and how to mitigate attacks from IoT devices have become critical, challenging issues.

Detecting and classifying malware is considered to be an essential step before further behavior analysis of malware to promote malware prevention and mitigation. However, CPU architecture diversity and IoT-device resource constraints render traditional signature-based protection methods unsuitable,

thereby hindering malware precautions and countermeasures. While dynamic analysis, executing binary programs in a sandbox environment to monitor their run-time behavior, is considered effective for Windows malware analysis, it suffers from difficulties in performing consistent analysis upon varying CPU architectures. Moreover, IoT devices with limited system resources typically do not support on-device dynamic analysis, thereby in turn rendering a malware protection method based on dynamic analysis unfeasible.

Consequently, existing research has been focusing on efficient approaches based on static analysis, such as reverse engineering of the binary programs of IoT malware [9]. In related work [10]–[12], experimental results with high detection rates were obtained by exploring the operation codes (opcodes) and control flow graphs (CFGs) of IoT malware. However, this related work did not take factors such as CPU architecture into consideration, and the evaluation datasets suffered from drawbacks such as limited scale and class imbalance, which can result in biased results. Moreover, these approaches are considered to be resource intensive, because opcode sequences can be obtained only from advanced reverse engineering tools.

In contrast to recent malware or applications on Windows and Android, current IoT programs tend to be implemented following common programming principles such as simplicity and straightforwardness – obfuscation and evasive techniques are used only rarely. Thus, we can take advantage of their simplicity to devise an efficient and effective method for detecting and classifying IoT malware. In this paper, we take the sequence of bytes beginning at the *entry point* as the input feature for machine learning methods. Generated by a compiler or a linker, the *entry point* of an executable file is the point at which the first instructions of a program are executed. Consequently, byte sequences from the entry point provide identifying information regarding the principal action and semantics of the program, which can be used to differentiate malware programs from benign programs (benignware). Furthermore, compared with the opcodes or CFGs, extracting the byte sequences from an executable file is straightforward and does not require advanced tools. Experiments have shown that we can achieve accurate malware detection and malware family classification based on a few hundred bytes beginning from the entry points.

To verify the performance of the proposed malware detection and family classification method across different CPU architectures, we created a large-scale multi-platform dataset consisting of IoT malware and benignware from a wide range of CPU architectures, including ARM, MIPS, x86, x86_64, PowerPC, SPARC, and Renesas SH. The dataset includes 111K benignware and 111K malware. The benignware samples were collected from commercial IoT-device vendors, including D-Link, Zyxel, Netgear, IDIS, Belkin, and MikroTik. The malware samples were downloaded from VirusTotal [13] and include the most recent malware from eight families, Mirai, Tsunami, Hajime, Dofloo, Bashlite, Xorddos, Android, and Pnscan.

The machine learning methods examined to detect and classify IoT malware include support vector machine (SVM), *k*-nearest neighbor (*K*NN), naive Bayes (NB), and multilayer perception (MLP). Experimental results show that the proposed method has promising generalization performance: more than 99.96% accuracy for malware detection and more than 98.47% accuracy for malware family classification, respectively. The results show that byte sequences from the entry point can serve as effective and efficient input features for IoT malware analysis. An efficient malware protection and mitigation solution can be implemented based on the findings in this paper.

The contributions of the paper are as follows.

- We propose using byte sequences at entry points of executable and linkable format (ELF) executable files as input features for IoT malware analysis;
- we provide a fast implementation of the *N*-gram method which is suitable for dealing with byte sequence features;
- we evaluate a number of popular machine learning algorithms on the feature set to demonstrate the efficacy and efficiency of the proposal;
- we create the largest dataset thus far for IoT malware analysis and compare the proposed method with related work;
- we reveal the necessity to perform CPU-specific analysis on IoT malware and provide solid results as proof of concept.

II. BACKGROUND AND RELATED WORK

In this section, we provide background information regarding IoT malware and review related work on IoT malware detection and classification with a focus on static analysis.

A. IOT MALWARE

Owing to commonly known vulnerabilities, such as weak, guessable, or hard-coded passwords, lack of secure update mechanisms, and insecure network services, IoT devices have become the most attractive targets of malware. By taking advantage of the vulnerabilities found on victimized devices, attackers can take control of the infected devices and force them to join the army of zombie devices known as the botnet. Consider *Mirai*, an infamous IoT malware, as an example. This malware scans vulnerable IoT devices and conducts brute-force login attacks using telnet or the Secure Shell (SSH) protocol, based on a predetermined dictionary of logins and passwords. Once login is successful, it downloads the malware executables and takes control of the victimized system [14] through a stepping stone commonly known as a command & control (C&C) server. After a botnet consisting of multiple infected devices is created, the attacker can conduct a distributed denial-of-service (DDoS) attack on any target through the C&C server. The release of the Mirai source code on GitHub in September 2016 led to a burst of attacks abusing the botnet, followed by the emergence of a large number of modified variants taking advantage of other vulnerabilities. For example, instead of taking advantage of

TABLE 1. Summary of Related Work

Citation	Feature	Algorithm	Accuracy	Dataset		Num. of Families	CPU Platform	Task
				Malware	Benignware			
[11]	Opcode	LSTM	98.18%	281	270	2	ARM-based IoT	detection
[12]	Opcode	AdaBoost	99.34%	247	269	2	ARM-based IoT	detection
[20]	Opcode	DBN	98%	4,600	4,600	2	Windows PE	detection
[21]	Opcode	SVM	98% (f-measure)	1,260	1260	2	Android	detection
[21]	Opcode	SVM	98% (f-measure)	1,260	-	49	Android	classification
[22]	CFG	CNN	99.66%	2,962	2,999	2	IoT	detection
[22]	CFG	CNN	99.32%	2,962	2,999	4	IoT	classification
[23]	PSI graph	DGCNN	92%	4,002	6,031	2	IoT	detection
[24]	Binary	CNN	93.33%	243	122	2	IoT	detection
[24]	Binary	CNN	81.8	243	122	3	IoT	classification
[25]	ELF header	Decision Tree	99.98%	709	734	2	IoT	detection
[26]	Byte sequences	Naive Bayes	97.11%	3,265	1,001	2	Windows PE	detection
[27]	Byte sequences	MDBA	94.67%	5,500	5,455	2	Windows PE	detection
Proposed method	Byte sequences	SVM	99.9%	111K	111K	2	IoT	detection
Proposed method	Byte sequences	SVM	98.4%	111K	-	8	IoT	classification

telnet or SSH, recent Mirai variants use known Remote Code Execution vulnerabilities to conduct attacks.

Mirai's source code is based on its predecessor, *Bashlite* or *Gafgyt*, first discovered in 2014. While they both make use of the vulnerabilities of known authentication credentials on IoT devices and share many similarities, Mirai improves on Bashlite on multiple fronts. See [14] for more information on these two botnets.

Another well-known malware family is Hajime, which was first identified by researchers in Rapidity Network [15]. Although some of the attack patterns of Hajime are similar to those of Mirai, they communicate with bots differently. Rather than using a C&C server, Hajime makes use of a poplar peer-to-peer (P2P) distributed hash table to download files [16].

Other well-known malware that target IoT devices include Dofloo and Xorddos. Dofloo, also known as Spike, is a backdoor malware that conducts DDoS attacks. It encrypts its commands using the Advanced Encryption Standard to avoid detection and analysis [17]. Xorddos is a recent botnet malware that targets Linux hosts on cloud systems. Recent research reports that new variants of Xorddos begin to target exposed Docker servers by searching for hosts with exposed Docker application programming interface (API) ports [18], [19].

B. MALWARE DETECTION AND MALWARE FAMILY CLASSIFICATION

Because traditional signature-based approaches cannot detect unknown malware, most security vendors today have adopted machine learning based solutions for malware detection and mitigation. Machine learning methods can learn the feature patterns from existing malware and use created models to detect novel malware with high efficiency and accuracy. In this section, we survey the most recent research on machine-learning based IoT malware analysis. The basic information of all of the reviewed methods is summarized in Table 1.

1) OPERATION CODE (OPCODE)

As the portion of machine language instructions that specifies the operations to be performed, opcodes are regarded as one of the common features for malware detection. Pajouh *et al.* [11] took the opcode sequences as distinguishing features in the long short-term memory (LSTM) algorithm and obtained 98.18% accuracy on a dataset consisting of 281 ARM-based malware and 270 ARM-based benignware. Darabian *et al.* [12] counted the number of occurrences of each opcode, and found that malware uses some specific opcodes more frequently than benignware. The experimental results showed that the classifier could reach an accuracy of 99% on ARM-based IoT samples. The classic N-gram representation has also been applied to opcodes to extract distinguishing string features. Ding *et al.* [20] extracted opcode N-gram features after using a disassembler for Windows portable executable (PE) [1] files and trained a deep belief network to detect malware with 98% accuracy. Similarly, Kang *et al.* [21] presented an approach based on opcode N-gram features for Android malware family classification using NB, SVM, partial decision tree (PART), and random forest. The experimental results showed that SVM can achieve a 98% F-measure in both malware detection and malware family classification.

2) GRAPH-BASED FEATURES

The most popular graph-based feature examined for malware analysis is CFG – a data structure that represents the order of opcode execution in a file. Alasmary *et al.* [22] extracted CFGs and characterized the executables by graph features including such features as numbers of nodes and edges, density, centrality, shortest path. Convolutional neural networks (CNNs) were then used for the analysis and yielded 99.66% accuracy for detection and 99.32% accuracy for malware family classification. Nguyen *et al.* [23] focused on the detection of IoT botnets by using printable string information (PSI) graphs as the primary feature for learning. They used a deep graph convolutional neural network (DGCNN) classifier and obtained an accuracy of 92%.

[1]Portable executable format is a file format for executables, object code, DLLs and others files used in 32-bit and 64-bit versions of Windows operating systems.

3) OTHERS

Su *et al.* [24] introduced a method for converting an ELF file into a gray-scale image and conducting learning on the image set. The algorithm first converts the file to a binary string, then combines the binary values into a vector of bytes, and finally transforms the vector into a gray-scale image. An accuracy of 93.33% on malware detection was obtained by applying deep learning to the images.

Shahzad *et al.* [25] extracted 383 structural features including section headers, symbol sections, and program headers from ELF files. Then, forensic analysis was used to sort and select useful features for learning. They evaluated the performance by using rule-based machine learning classifiers such as repeated incremental pruning to produce error reduction (RIPPER) and obtained accuracy values greater than 99%.

Although these machine learning based solutions were able to achieve a high accuracy in malware detection and classification, they did not consider an essential feature of IoT executables, the diversity of CPU architectures. Since the instruction sets used on different CPU architecture are different, knowledge regarding an executable file compiled on one CPU architectures provides little generalization information for one compiled on another CPU architecture, even if they are based on the same source file. Therefore, analysis on a dataset consisting of executables from different CPU architectures is typically not effective. While the dimension of the feature space is significantly increased to cover a superset of opcodes from all CPU architectures, little gain in generalization performance can be expected.

In this sense, graph-based features that capture the high-level characters of software behavior have the potential to provide cross-platform generalization ability. Nevertheless, they can also suffer from other drawbacks. First, as the generalization performance typically depends on the granularity of the characterizing features, high-level but coarse features typically do not yield better results than low-level refined features. Second, extracting graphs from executables requires a consistent tool across all CPU platforms, and this is never easy to attain. Finally, the computing resources for obtaining graph properties when the graph is large might not be available on IoT devices with limited system resources.

C. BYTE SEQUENCES IN STATIC ANALYSIS

An executable file, or executable, commonly takes the form of a binary file composed of sequences of bytes that contain information regarding commands and data that can be interpreted by a machine to perform specific functions. Schultz *et al.* [26] used byte sequences, DLL calls, and strings as features in a machine learning method to detect PE malware and achieved an accuracy of 97.11%. Li *et al.* [27] presented a PE-malware detection method based on association rule mining. They used N-grams to represent segments of byte sequences and reported an accuracy of 94.67%. Ban *et al.* [28] extracted a finite-length byte sequence at the entry point of an executable to perform packer identification.

D. ENTRY POINT

Generated by the compiler, an entry point of an executable represents the starting execution address of a program. The executable file format of Linux is ELF and Listing 1 shows the structure of a 32 bit ELF header, where the entry point is defined in the `e_entry` field.

Listing 1 The structure of 32 bit ELF header

```
typedef struct elfhdr {
    unsigned char  e_ident[EI_NIDENT];
    Elf32_Half    e_type;
    Elf32_Half    e_machine;
    Elf32_Word    e_version;
    Elf32_Addr    e_entry;
    Elf32_Off     e_phoff;
    Elf32_Off     e_shoff;
    Elf32_Word    e_flags;
    Elf32_Half    e_ehsize;
    Elf32_Half    e_phentsize;
    Elf32_Half    e_phnum;
    Elf32_Half    e_shentsize;
    Elf32_Half    e_shnum;
    Elf32_Half    e_shstrndx;
} Elf32_Ehdr;
```

Consider a C program as an example. Typically, a program is executed from the *main()* function, and GNU Compiler Collection (gcc) uses `_start` symbol as an entry point to initialize the *main()* function. However, a different entry point address can be assigned intentionally during compilation. Therefore, the entry point of one program compiled by an author is likely different from that of a program created by another author. The entry point also changes when an executable is packed and encrypted. Although packers are well developed and commonly used as an obfuscation technique with Windows PE files, there have been very few cases observed as with IoT executables.

III. METHODOLOGY

In this section, we describe the details of the proposed method. Byte sequences obtained from executable files are taken as the input features for machine learning algorithms for performing IoT malware detection and family classification. Fig. 1 shows an overview of our method which consists of three parts: collecting ELF files (samples in the dataset) from different sources, feature extraction and labeling for each sample, and model training and evaluation following a standard pipeline of machine learning processing.

A. DATA COLLECTION

According to a recent survey performed through Eclipse [29], Linux OS has become the dominant OS for IoT gadgets, working on more than 80% of the market. Consequently, Linux OS has become the major target of IoT-oriented malware programs. Malware arrives at IoT devices in the form of

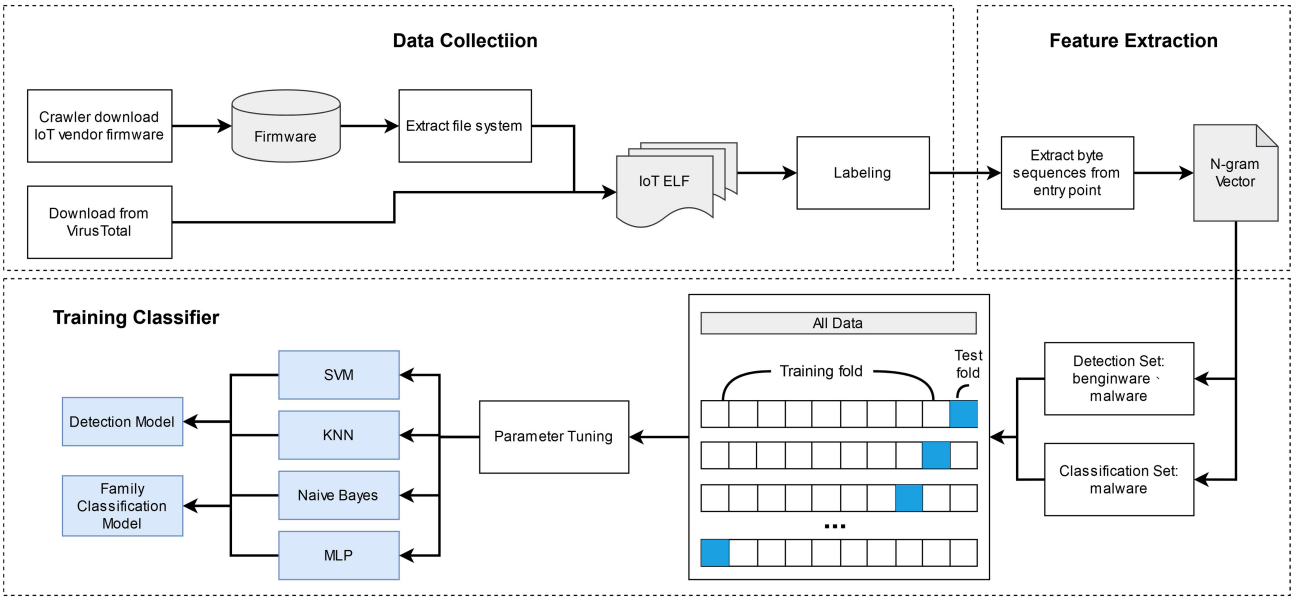


FIG. 1. **Overview of IoT malware analysis framework.**

compiled ELF files, the standard binary file format for Unix and Unix-like systems. Designed to be flexible, extensible, and cross-platform, ELF can be used by different operating systems on different hardware platforms.

We collect benign ELF files from officially released IoT firmware images. As these executables are provided by IoT vendors of Linux-based systems, they are taken as benignware and are labeled negative in the dataset. The procedure of collecting benignware is as follows:

1) Firmware images are downloaded from various IoT vendors including D-Link, Zyxel, Netgear, IDIS, Belkin, and MikroTik. During this process, customized downloading scripts are created to facilitate automated downloading from different IoT vendors.
2) After firmware images are collected, *binwalk* [30] – a command that checks the file type – is used to extract the ELF files from system locations such as "`/bin`", "`/etc`", "`/dev`", "`/home`", "`/lib`", "`/mnt`", "`/opt`", "`/root`", "`/run`", "`/sbin`", "`/tmp`", "`/usr`", and "`/var`".
3) Duplicated files are removed from the collection by checking their uniqueness using hash functions such as Secure Hash Algorithm 1 (SHA1).
4) These files are sent to VirusTotal and examined by different antivirus vendors to verify their class labels.

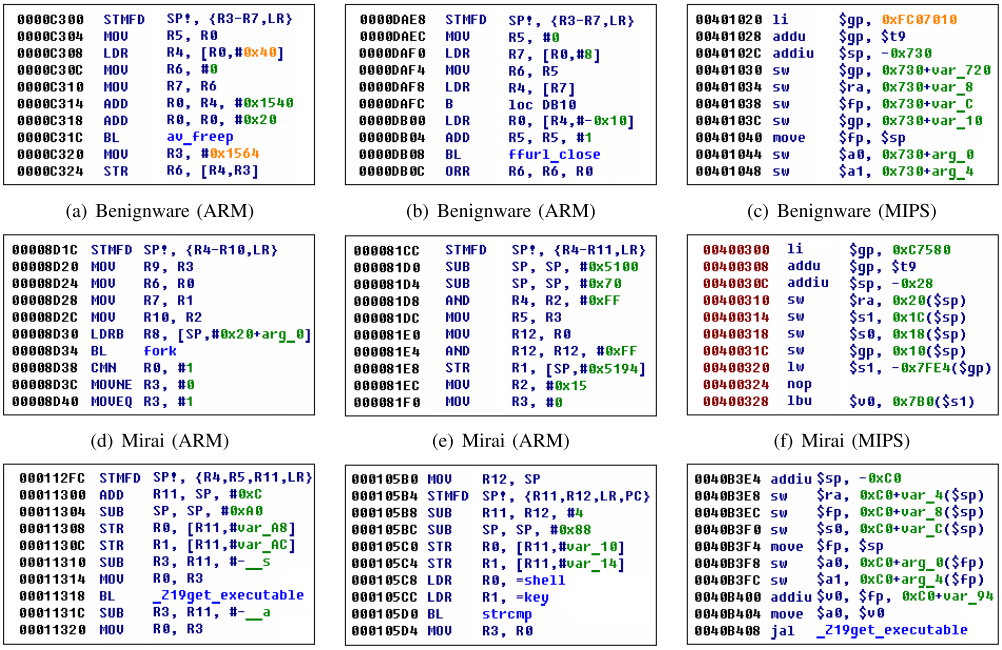
We collect malicious ELF files from VirusTotal [13] – a website that aggregates many antivirus products and online scanning engines to check for viruses that might be missed by a single antivirus engine. The malware samples belong to eight families including Mirai, Tsunami, Hajime, Dofloo, Bashlite, XORDDoS, Android, and Pnscan. The family name of a sample is decided by majority vote of the antivirus reports from VirusTotal. For example, if eight antivirus engines label a particular malware sample as Mirai, and two label it as Tsunami, then we label the sample Mirai. A sample is removed from the dataset if there is a tie or if fewer than five engines vote for the winning label.

Previous work on binary file analysis has revealed that the binary code at the entry point of packed binaries contains basically information regarding the packer [28], [31]. For a packer-obfuscated ELF file, the entry point no longer contains run-time information regarding the original entry point of the program. To retain the coherence of the dataset, we remove the packed samples detected by Radare2 [32].

B. FEATURE EXTRACTION

The entry point indicates the position which the processor enters a program or a code fragment and begins execution. Since executables generally perform different operations at the starting point, we use the byte sequences at entry points to detect whether an ELF file is malware. As Fig. 2 shows, executables created from different sources generally differ drastically. For a program that is subject to modification, for example, subvariants of the same malware, the code sequence tends to be highly similar to the original as long as the source code that corresponds to the starting section of the executables is not totally rewritten.

The procedure for creating a numerical feature vector from a byte sequence extracted at the entry point is as follows. First, we locate the entry point of an ELF file by checking the `e_entry` field of the ELF header. Second, we use *pwntools* [33] to take the first L bytes starting from the entry point as the input of the algorithms. Choosing an appropriate L value enables us to ignore irrelevant information such as data segments following the code segment. Moreover, the L value also helps to balance the quantity of information explored in the code related to the computation complexity of the learning. For a file that is not sufficiently long, the byte sequence is



(a) Benignware (ARM) (b) Benignware (ARM) (c) Benignware (MIPS)

(d) Mirai (ARM) (e) Mirai (ARM) (f) Mirai (MIPS)

(g) Dofloo (ARM) (h) Dofloo (ARM) (i) Dofloo (MIPS)

FIG. 2. Opcode sequences extracted at the entry points of different executables.

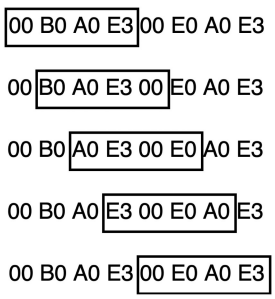


FIG. 3. Example of extracting 4-grams from a byte sequence ($L = 10$).

zero-padded to length L. Then, we follow the feature extraction conventions for N-gram models. Typically, N-grams are used in natural language processing to build learning models based on contiguous sequences of length N. An N-gram in our setting represents a substring (contiguous bytes) of length N extracted from the original byte sequence. Fig. 3 shows an example of extracting 4-gram features from a byte sequence.

Based on the N-gram representation, an executable can be formulated as a numerical vector, where each dimension of the vector corresponds to a specific N-gram and the value along the dimension is the frequency of the N-gram in the byte sequence. Owing to the extraordinarily high dimension of all N-grams occurring in the dataset, we use a sparse representation to reduce the memory consumed during learning.

C. CLASSIFICATION METHODS

The sparsity and high dimensionality of the N-gram representation render it intractable for many machine learning algorithms. In this section we select four classification methods that can deal efficiently with high dimensional sparse data, SVM, KNN, NB, and MLP. We use these classifiers to build prediction models for the dataset created under the process introduced in Section III-B.

SVM is a supervised learning model that finds a hyperplane with maximized margin to distinguish samples from two classes. We use SVM to solve a multi-class classification problem following the one-against-all convention: For an m-class problem, we first construct m binary SVM classifiers, each of which separates one class from the rest. Then, we determine the predicted label by a majority vote of all of the classifiers.

KNN is a popular classification algorithm – a sample is classified by a majority vote of its k-nearest neighbors. The results of KNN indicate well how much discriminant information can be captured in the similarity function defined on the feature vectors. As generalization is deferred until a query sample is known, the computational cost of KNN in prediction (depending primarily on nearest neighbor search for the query sample) tends to be intensive even for datasets with moderate sample sizes.

NB is a classifier based on Bayes' theorem. It assumes that all of the features are statistically independent – the value of a particular feature is independent of the value of any other feature. While requiring that the number of parameters be linear in the number of features, NB classifiers are highly scalable, a fact that satisfies the requirements of our learning task well.

MLP is a class of feedforward artificial networks consisting of at least three layers, an input layer, a hidden layer, and an

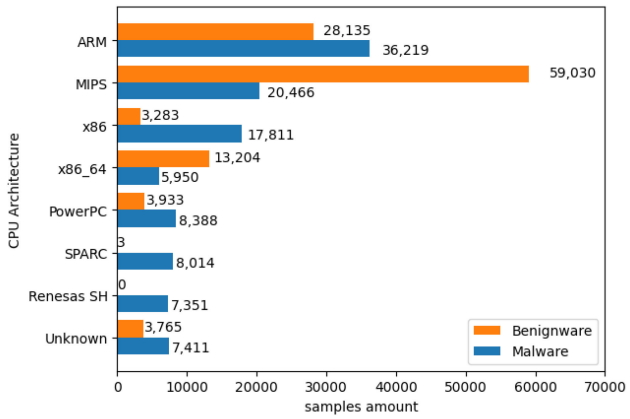


FIG. 4. Sample distribution among different CPU architectures.

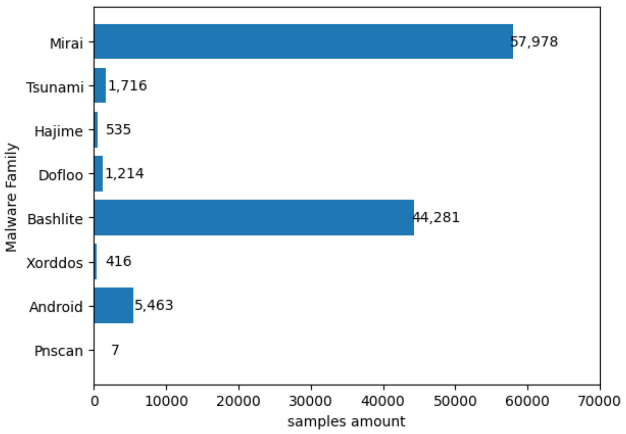


FIG. 5. Sample distribution among different malware families.

output layer. MLP makes use of a supervised learning technique called backpropagation for training. In our experiment, we used two hidden layers of 50 nodes each.

IV. EXPERIMENT

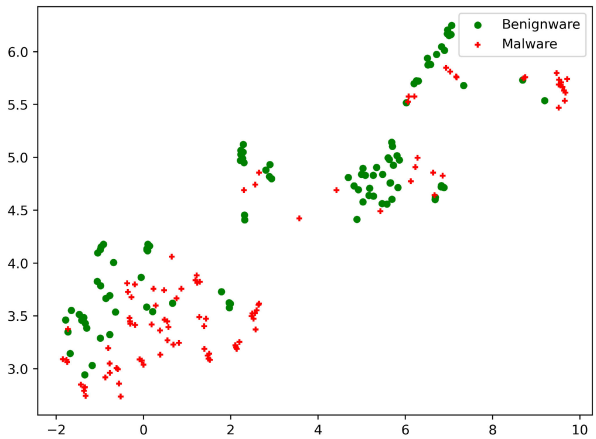
We evaluated the performance of SVM, KNN, NB, and MLP on two learning tasks: malware detection, which aims to differentiate malware executables from benign ones, and malware family classification, which aims to predict the category of a malware executable among known families. Our experiments are based on *scikit-learn* [34] implemented in Python 3.7.

A. DATASET

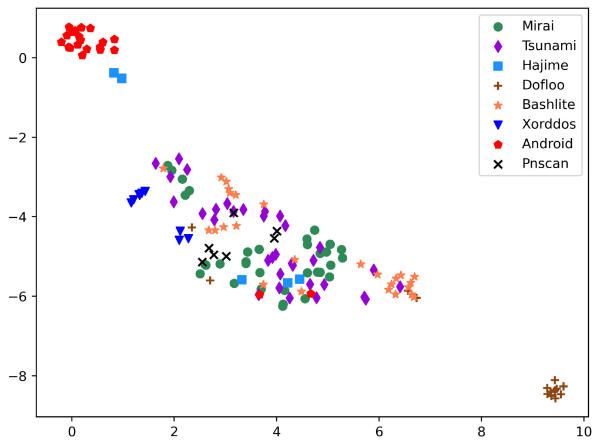
Following the process introduced in section III-A, we collected 222,963 ELF executable files from IoT devices. In the dataset, 111,353 files are benignware and 111,610 are malware. The distribution of the samples among different CPU architectures is shown in Fig. 4. As shown in the figure, the two most popular CPU architectures for malware are ARM and MIPS. We failed to collect sufficiently many benignware samples for Renesas SH and SPARC: no firmware images are publicly available for Renesas SH, and most of the firmware images of SPARC are encrypted. Fig. 5 shows the distribution of the malware executables among different malware families. The numbers of Mirai and Bashlite samples are much greater than the others because of the release of source code.

B. VISUALIZATION

To enhance understand of the data distribution, we use uniform manifold approximation and projection (UMAP) [35] to visualize the data in a two-dimensional (2-D) embedding space. UMAP is a nonlinear dimension reduction method that can be used to approximate the relationships among high-dimensional data in a low-dimensional layout. It is computationally efficient and can deal with large high-dimensional datasets. Fig. 6(a) shows the 2D visualization result of a subset of 100 benignware samples and 100 malware samples with $L = 512$ and $N = 4$. In the figure, a green point represents a benignware sample and a red point represents a malware sample. We can see that despite the low dimension of the layout, there is high separability between benignware samples and malware samples: a benignware sample falls close to other benignware samples but apart from malware samples. High



(a) A visualization of benginware and malware ($L = 512, N = 4$)



(b) A visualization of malware family ($L = 1024, N = 4$)

FIG. 6. UMAP 2D visualization of sample distribution.

TABLE 2. Settings for Parameter Tuning

Parameter	Classifiers	Physical Meaning	Grid Values
N	SVM, KNN, NB, MLP	Parameter N as for N-gram	{3, 4, 5, 6, 7}
L	SVM, KNN, NB, MLP	Input byte length starting at entry point	{128, 256, ..., 1024}
C	SVM	Penalty parameter	$\{2 \cdot 10^{-5}, 2 \cdot 10^{-4}, ..., 2 \cdot 10^{12}\}$
K	KNN	Number of nearest neighbors	{1, 3, ..., 9}

separability implies good prediction performance for malware detection.

The distribution of a subset of 200 malware samples among different malware families is shown in Fig. 6(b). In the figure, each unique color indicates a different malware family. In contrast to the previous figure, except for the two malware families, Android and Tsunami, there are no clear boundaries that can separate samples from different malware families in the 2D embedding. This indicates that malware family classification is more difficult: as samples from different malware families are intermingled, there is a greater chance for a classifier to make incorrect predictions.

C. PARAMETER TUNING

As is commonly known, the generalization performance of learning algorithms relies heavily on the hyperparameters used to train the model. Given a training set A and a test set B, we tune the parameters of the algorithms to be used following the procedure below, using grid search. For each of the parameters of a given model, we define a series of values, i.e., grid values, then all possible combinations of grid values on different parameters form a pool of parameters settings. From the pool, the parameter setting with the best classification result is determined using 10-fold cross validation. First, A is shuffled randomly and split evenly into ten groups, $A_i (i = 1, \ldots, 10)$. For the i-th run, A_i is taken as the testing set while the rest of the data, $A \backslash Ai$, is used as the training set. Then the classification model is trained based on $A \backslash Ai$ and evaluated on A_i. For each parameter setting, this process is repeated 10 times over $A_i (i = 1, \ldots, 10)$ and the average accuracy is used to measure the classification performance of the parameter setting. Table 2 lists the grid values of all the parameters that we have examined for each classifier.

As an example of parameter tuning, Fig. 7 shows the tuning process of the length parameter, L, for malware detection using SVMs. The blue line depicts the overall prediction accuracy averaged from all of the combinations of (N, C) for each L parameter. The green line shows the average training time used to build the models with each L parameter. From the graph, we can see that the accuracy first grows as L increases from 128 to 384; and begins to drop after L surpasses 640. At the same time, the training time continues rising as L increases. The best accuracy is obtained at $L = 384$ and $L = 640$. However, the training time at $L = 640$ almost doubles that at $L = 384$. This result suggests that 384 is the most suitable input byte-length value for SVMs.

Fig. 8 shows the process of tuning parameter N for SVMs. Lines represent the averaged accuracy obtained with different N values (from three to seven in the graph). According to the

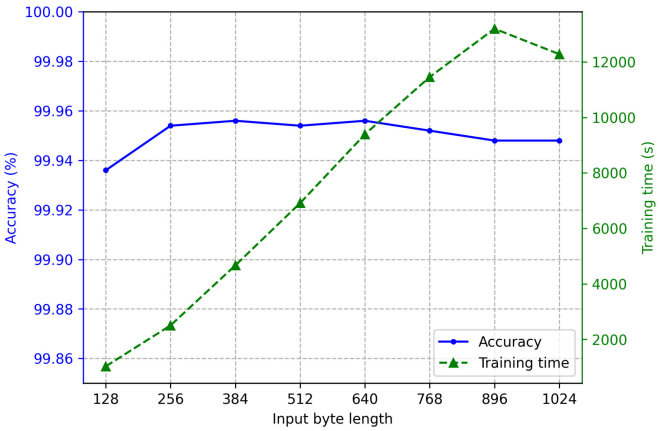


FIG. 7. Tuning parameter *L* using 10-fold cross validation for SVM. *L* is selected from {128, 256, 384, 512, 640, 768, 896, 1024}.

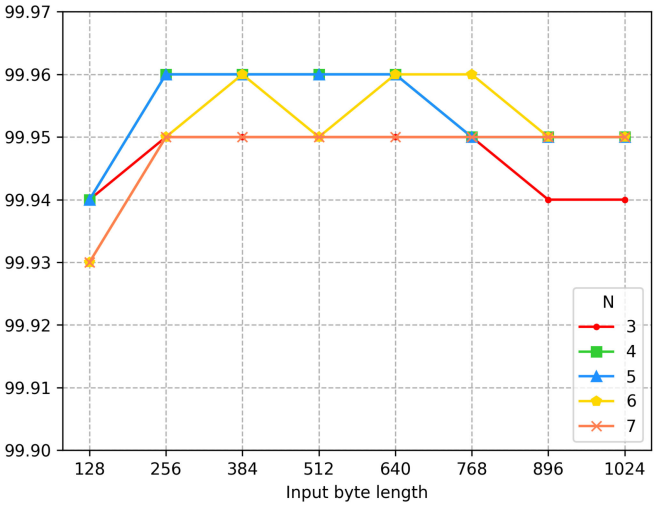


FIG. 8. Tuning parameter *N* using 10-fold cross validation. *N* is selected from {3, 4, 5, 6, 7}.

previous result on tuning the L parameter, $L = 384$ gives the best result. With this setting, $N = 4$, $N = 5$, $N = 6$ yield the same highest accuracy. We chose $N = 4$ for SVMs in later evaluation as it provides a feature space with far fewer dimensions and thus consumes less training and prediction time.

D. EVALUATION METRICS

The parameter setting with the best classification result is selected from the pool to train an overall model using A and then the model is evaluated on test set B. We make use of widely used metrics including accuracy, precision, recall, and

TABLE 3. Evaluation Results for Malware Detection

Classifier	Accuracy(%)	Training Time(s)	Testing Time(s)	Parameters
SVM	99.96	3,466	0.0006	$L = 384, N = 4, C = 1$
KNN	99.73	772	0.526	$L = 128, N = 4, K = 3$
NB	99.73	1,679	0.097	$L = 768, N = 7$
MLP	99.89	3,326	0.0049	$L = 512, N = 4$

TABLE 4. Evaluation Results for Malware Family Classification

Classifier	Accuracy(%)	Training Time(s)	Testing Time(s)	Parameters
SVM	98.47	13,475	0.0004	L=1024, N=4, C=0.2
KNN	98.18	443	1.4	L=768, N=4, K=3
NB	96.76	433	0.64	L=768, N=7
MLP	97.87	4,496	0.0049	L=1024, N=4

false positive rate (FPR) to measure the generalization performance of the classifiers. These metrics are defined based on the following intermediate measures.

- True positive (TP): samples classified correctly as positive;
- False positive (FP): samples classified incorrectly as positive;
- True negative (TN): samples classified correctly as negative;
- False negative (FN): samples classified incorrectly as negative.

Accuracy is the probability of test samples being classified correctly:

$$Accuracy = \frac{TP + TN}{M}, \tag{1}$$

where M is the total number of samples used for evaluation. Precision is the probability of predicted positives being classified correctly:

$$Precision = \frac{TP}{TP + FP}. \tag{2}$$

Recall is the probability of the samples in the positive class being classified correctly:

$$Recall = \frac{TP}{TP + FN}. \tag{3}$$

FPR is the probability of negative samples being assigned incorrectly to the positive class:

$$FPR = \frac{FP}{FP + TN}. \tag{4}$$

E. NUMERICAL RESULT

We design two experiments to demonstrate the performance of the proposed approach. The first experiment is conducted on an overall dataset consisting of all of the samples in our dataset. In the second experiment, the evaluation is conducted on eight subsets formed by taking all samples from the same CPU architecture respectively. See Fig. 4 for information regarding the sizes of the subsets. Following the procedure introduced in section IV-C, we chose the best parameter settings for each classifier independently. The selected parameter settings are shown in the rightmost column in Tables 3 and 4. We report the evaluation results based on 10-fold cross validation.

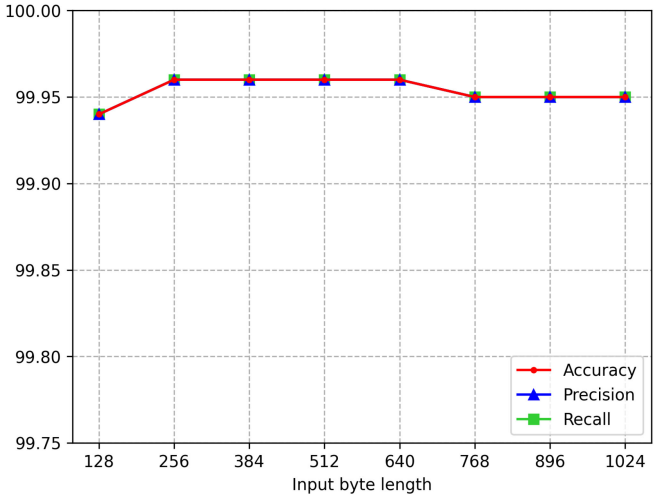


FIG. 9. Malware detection results using SVM with $N = 4$.

That is, results in the tables are the average numbers over ten independent runs with the training and test sets determined by 10-fold cross validation.

1) IOT MALWARE DETECTION

The malware detection experiment aims to evaluate the discriminating performance of the proposed method in differentiating malware samples from benignware samples. The typical scenario involves raising some form of security alert to warn the user when a downloaded program is recognized as malware. Table 3 shows the evaluation results of all examined classifiers. All of the classifiers yield near optimal results with accuracy greater than 99%, indicating that the byte sequences at the entry point carry essential information for differentiating malware from benignware on all IoT platforms. SVM provides the best accuracy of 99.96% with a slightly longer training time than other methods. In addition, it requires the least time for making a prediction. SVM requires only a D dimensional vector and a bias parameter to make a prediction, where D is the dimension of the feature vectors. Its high generalization performance and efficiency in prediction make it the most suitable for IoT devices.

Taking SVM as an example, Fig. 9 shows how input byte length affects the performance of the classifier. The best accuracy is retained when L varies from 256 to 640, and the accuracy drops when L is set to values outside of this range. This result indicates that for IoT executables, the first 300 to 600 bytes carry the most discriminating information to differentiate malware from benignware. A value less than 300 might lead to information loss and a value greater than 600 might introduce information extraneous to the data. In this task, because of the near optimal performance of the algorithm, the precision and recall on the malware class are very close to the accuracy values, shown as overlapped lines in the figure.

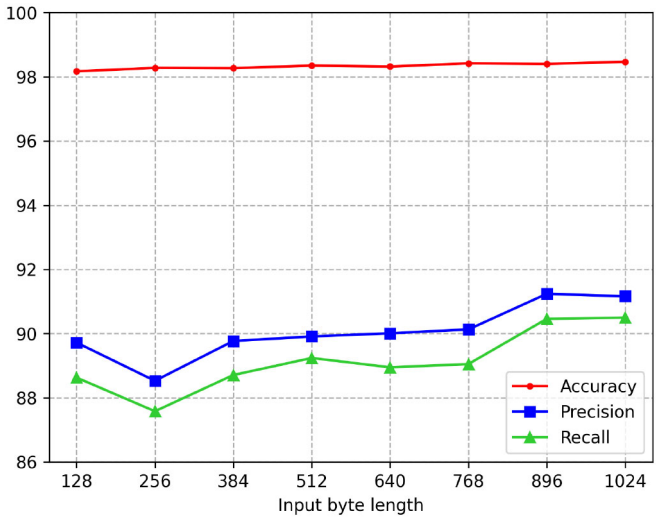


FIG. 10. Malware family classification results using SVM with $N = 4$.

Using a shorter byte length, *K*NN yields 99.73% accuracy, which is a bit less than SVM, using a training time (to build an indexing data structure to support fast nearest neighbor search) 20% of SVM's. *K*NN must store all of the training data together with the indexing data structure in the main memory to support fast nearest neighbor search. It also requires more time to make a prediction than SVM.

With slightly longer input byte lengths, NB and MLP produce results slightly inferior to those of SVM. These two algorithms require only moderate memory to store models for prediction, a fact that is also reflected in the testing time cost.

2) FAMILY CLASSIFICATION

Malware family classification aims to predict the category of a detected malware among the listed known families. Information regarding the malware family can enable the user to apply appropriate mitigation strategies for preventing malware infection or reducing damage from the malware. The malware family classification results of all of the classifiers are shown in Table 4. SVM achieves the highest accuracy of 98.47%, while *K*NN and MLP show slightly lower accuracies. NB yields an accuracy of 96.76% which is the lowest for the four classifiers.

Fig. 10 shows the generalization performance of SVM classifiers obtained with different input byte lengths. The best performance is obtained at $L = 1024$: the highest accuracy and recall, and the second highest precision, which is slightly less than the precision at $L = 896$. Inferior performance is obtained when L is set to smaller values: When $L = 256$, precision and recall are both less than 89%. The comparatively longer input byte length required for malware family classification to obtain the best generalization performance demonstrates the greater complexity in malware family classification than in malware detection: more distinguishing information is required to differentiate malware from different families than benignware.

3) CPU-SPECIFIC RESULTS

In this experiment, we divided the dataset into eight subsets based on the supported CPU architectures of the samples. Since there are insufficiently many benignware samples on SPARC and Renesas SH to form a proper dataset, we omit the experiments on these two architectures.

Fig. 11 shows the data distribution in the embedded 2D space, with each unique color representing a different CPU architecture. To enhance the readability of the figure, we use a subset consisting of 400 samples of the dataset for visualization purposes. We have confirmed that using UMAP on the full dataset produces a layout similar to the one shown in the figure. We can see that samples from the same CPU architecture form tight clusters. Within the clusters, benignware samples and malware samples, as denoted respectively by colored disks and plus signs, still show clear separability. This result encourages us to pursue a CPU-specific classification.

Table 5 shows the results of malware detection and family classification among specific CPU architectures. We report only the results of SVM and *K*NN in the table to enhance readability. NB and MLP show generalization performances comparable to those of SVM and *K*NN.

For malware detection, SVM ($L = 256$ and $N = 4$) reaches accuracy values greater than 99.96% on all CPU architectures except on x86_64. With the same parameter settings, *K*NN yields accuracy values greater than 99.73% except on x86_64. Note that 99.96% and 99.73% are the accuracy values obtained using the overall dataset. These results show that CPU-specific analysis can further improve the generalization performance of the classifiers.

For malware family classification, SVM ($L = 1024, N = 4$) yields accuracy values greater than 98.47% on four out of eight CPU architectures, ARM, SPARC, Renesas SH, and Unknown. For three of the four remaining CPU architectures, MIPS, X86, and PowerPC, SVM produces accuracy values very close to 98.47%. The CPU-specific analysis provides inferior performance only on x86_64. *K*NN also yields accuracy values greater than or very close to 98.15% on seven of the eight CPU architectures.

Performance of malware detection and malware family classification on x86_64 is always slightly lower than on other platforms. A possible explanation for this result is that the executables collected for the x86_64 architecture might include files from a wide range of Linux distributions, rendering the classification tasks more complicated.

We compare the results of the two scenarios in Table 6. Numbers in the table are the weighted averages of the numbers reported in Table 5 according to the following equation:

$$\text{Weighted Metric} = \frac{1}{M}\sum_{i=0}^{C} m_i \cdot \text{Metric}(i), \tag{5}$$

where C is the number of CPU architectures in the experiment, M the total number of samples in the dataset, and m_i the number of samples on the i-th CPU architecture. According to the malware detection results in the table, the weighted average

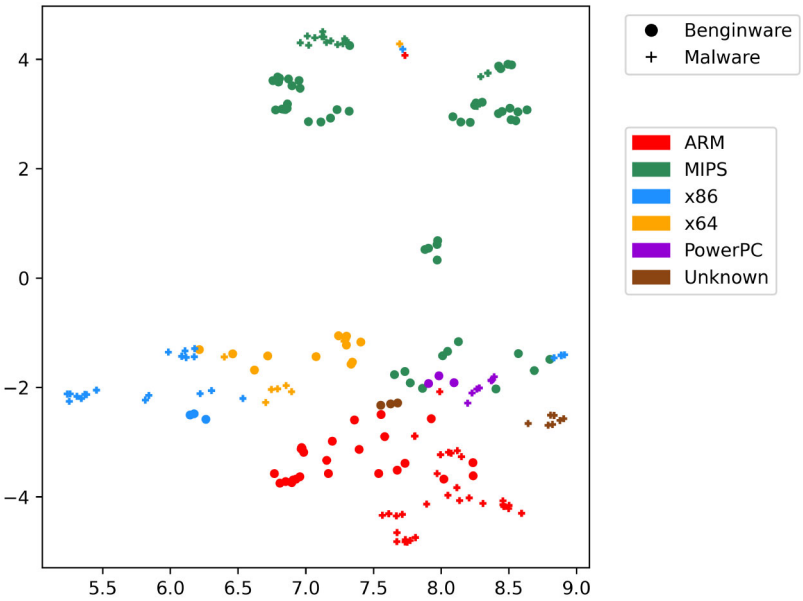


FIG. 11. **UMAP 2D visualization of sample distribution: benignware and malware on different CPU architectures.**

TABLE 5. **Results of CPU-specific Analysis**

Dataset	Classifier	Malware Detection ($L = 384, N = 4$)					Malware Family Classification ($L = 1024, N = 4$)				
		Accuracy(%)	Precision(%)	Reacll(%)	FPR(%)	Time(s)	Accuracy(%)	Precision(%)	Reacll(%)	FPR(%)	Time(s)
ARM	SVM	99.96	99.96	99.96	0.04	278	98.50	98.95	98.15	0.46	3,929
	KNN	99.90	99.90	99.90	0.1	172	98.08	95.18	93.83	0.06	203
MIPS	SVM	99.98	99.98	99.97	0.02	189	98.35	97.55	97.90	0.53	1,186
	KNN	99.94	99.92	99.92	0.08	380	98.08	97.50	96.63	0.61	53
x86	SVM	99.98	99.99	99.94	0.06	50	98.37	90.69	90.60	0.42	1,404
	KNN	99.96	99.90	99.94	0.06	20	98.38	91.45	90.05	0.44	53
x86_64	SVM	99.87	99.90	99.81	0.18	56	97.43	90.72	90.4	0.76	528
	KNN	99.72	99.69	99.65	0.35	14	97.24	91.52	91.55	0.91	15
PowerPC	SVM	99.98	99.96	99.98	0.02	41	98.41	96.15	94.13	1.08	398
	KNN	99.95	99.94	99.94	0.06	13	98.43	93.46	91.66	1.05	18
SPARC	SVM	NaN	NaN	NaN	NaN	NaN	99.0	98.88	93.89	0.59	252
	KNN	NaN	NaN	NaN	NaN	NaN	98.86	96.86	94.37	0.74	14
Renesas SH	SVM	NaN	NaN	NaN	NaN	NaN	98.61	97.24	96.48	0.96	374
	KNN	NaN	NaN	NaN	NaN	NaN	98.52	96.73	94.87	1.06	13
Unknown	SVM	99.96	99.95	99.97	0.03	57	99.04	95.57	93.05	0.67	503
	KNN	99.94	99.92	99.94	0.06	11	98.95	91.98	90.94	0.66	11

TABLE 6. **Result Comparison in Two Scenarios**

	Detection				Family Classification			
	All Data		Divide CPU		All Data		Divide CPU	
Classifier	SVM	KNN	SVM	KNN	SVM	KNN	SVM	KNN
Accuracy	99.93%	99.73%	99.96%	99.91%	98.47%	98.18%	98.47%	98.25%
Precision	99.93%	99.73%	99.96%	99.89%	90.00%	89.25%	96.38%	94.7%
Recall	99.94%	99.73%	99.95%	99.89%	89.09%	88.26%	95.43%	93.37%

of the accuracy values from CPU-specific SVM models is 99.96%, same as the 99.96% obtained from the overall SVM model on the full dataset. A similar conclusion applies to the precision and recall metrics. Moreover, the CPU-specific experiments of family classification yields an accuracy value of 98.47% which is identical to the result obtained from the overall model on the full dataset. However, this time we obtain much greater precision and recall values. A similar result can be observed on the results from the *K*NN classifiers. The improved performance using SVM and *K*NN confirms that information on CPU architectures can further improve the generalization performance of the classifiers both on malware detection and malware family classification.

TABLE 7. Performance Comparison With Related Work

Research	Feature extracting time (one file)	Feature	Classifier	Malware Detection			Family Classification		
				Accuracy	Training time	Testing time	Accuarcy	Training time	Testing time
Kang [21]	2m40s	opcode	SVM	97.87%	11m37s	0.00133s	94.41%	47m45s	0.0037s
			KNN	98.22%	1m28s	1.134s	96.08%	58s	0.85s
			NB	83.17%	21s	0.1147s	61.65%	28s	0.26s
			MLP	99.12%	16m55s	0.001s	96.83%	12m16s	0.0012s
Shahzad [25]	0.062s	ELF header	SVM	96.94%	1m29s	0.0029s	82.94%	2m15s	0.0023s
			KNN	90.87%	12s	0.0036s	80.28%	3s	0.0029s
			NB	72.6%	2s	0.0027s	29.3%	1s	0.0036s
			MLP	95.2%	6m15s	0.0014s	79.69%	3m05s	0.0019s
Proposed method	0.036s	byte sequences from entry point	SVM	99.53%	3m52s	0.0006s	96.44%	10m45s	0.00041s
			KNN	99.12%	7s	0.137s	95.9%	11s	0.2626s
			NB	98.49%	8s	0.010s	93.15%	17s	0.045s
			MLP	99.54%	4m57s	0.002s	96.48%	7m13s	0.0011s

V. DISCUSSION

A. COMPARISON WITH RELATED WORK

In this experiment, we compare the proposed approach with those in related work on malware detection and malware family classification. As reviewed in section II-B, different types of features are explored in the literature in connection with IoT malware analysis. We implement the opcode-based method introduced by Kang *et al.* [21] and the method proposed by Shahzad *et al.* [25] which makes use of information extracted from ELF headers. Because the cited methods suffer from scalability problems, we use a subset of 20K ELF files consisting of 10K malware and 10K benignware for evaluation.

The results shown in Table 7 of each classifier include time cost for feature extraction, accuracy, training time, and testing time. Although the time cost for feature extraction and training for the method in [25] is the fastest, its generalization performance is the worst on both malware detection (96.94% accuracy using SVM) and malware family classification (82.94% accuracy using SVM). This result indicates that the information in ELF headers contains considerable discriminating information for malware detection, but that is not enough to differentiate malware families. The accuracy of the method in [21] performs as well as the proposed method, but it requires much more time to extract features from files and to perform model training. The heavy costs in time and system resources render methods based on opcode features not suitable for IoT devices with limited resources.

The proposed method yields the highest accuracy on malware detection (99.54% using MLP) and the second highest accuracy on malware family classification (96.48% using MLP). Moreover, the time cost for feature extraction is only 0.036s. These results show that the proposed method is more efficient and effective than existing methods.

B. LIMITATION ON MALWARE FAMILY CLASSIFICATION

In section IV-E, we observed that malware family classification is inherently more difficult than malware detection. To investigate further which malware family is the most difficult to predict correctly, we examine the confusion matrix that stores the misclassified samples among different malware families. Fig. 12 shows the average number of misclassified samples in the confusion matrices obtained from 10-fold cross validation. We can see that most of the samples other than a portion of Mirai and Bashlite samples are classified correctly. This indicates that the byte sequences from entry points do not provide sufficient discriminating information for a portion of the malware from these two classes – a problem caused directly by the shared source code of the two types of malware. More bytes from the ELF file or other features such as opcodes might help to improve the performance of a malware family classifier.

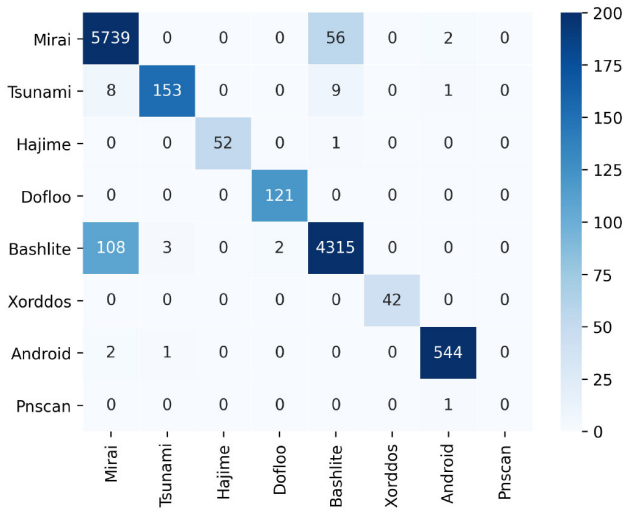


FIG. 12. Confusion matrix for malware family classification.

VI. CONCLUSION

In this paper, we proposed a novel approach for detecting IoT-oriented malware and classifying their families based on

the byte sequences extracted from ELF files. The experimental results on malware detection showed that a near optimal accuracy greater than 99.96% could be obtained using the proposed approach, and the performance can be improved further using CPU-specific models. Our method can also determine the malware family with greater than 98.47% accuracy. The results show that the proposed method outperforms existing solutions in terms of efficacy and efficiency. We believe that promising solutions for IoT security can be developed based on the findings of this paper.

REFERENCES

[1] M. Hung, "Leading the IoT," Accessed: Oct. 10, 2020. [Online]. Available: https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf

[2] G. D. Maayan, "The IoT rundown for 2020: Stats, risks, and solution," Accessed: Jul. 14, 2020. [Online]. Available: https://securitytoday.com/articles/2020/01/13/the-iot-rundown-for-2020.aspx

[3] S.-M. Cheng, P.-Y. Chen, C.-C. Lin, and H.-C. Hsiao, "Traffic-aware patching for cyber security in mobile IoT," *IEEE Commun. Mag.*, vol. 55, no. 7, pp. 29–35, Jul. 2017.

[4] J. Granjal, E. Monteiro, and J. S. Silva, "Security for the Internet of Things: A survey of existing protocols and open research issues," *IEEE Commun. Surveys Tut.*, vol. 17, no. 3, pp. 1294–1312, Jan. 2015.

[5] F-Secure, "Attack Landscape H2 2019," , Accessed: Jul. 14, 2020. [Online]. Available: https://blog-assets.f-secure.com/wp-content/uploads/2020/03/04101313/attack-landscape-h22019-final.pdf.

[6] Anna-senpai, "Mirai source code," Accessed: Oct. 10, 2020. [Online]. Available: https://github.com/jgamblin/Mirai-Source-Code/

[7] W. Li, Z. Su, K. Zhang, A. Benslimane, and D. Fang, "Defending malicious check-in using big data analysis of indoor positioning system: An access point selection approach," *IEEE Trans. Netw. Sci. Eng.*, pp. 1–1, Aug. 2020.

[8] A. Costin and J. Zaddach, "IoT malware: Comprehensive survey, analysis framework and case studies," in *Blackhat USA*, Aug. 2018.

[9] Q.-D. Ngo, H.-T. Nguyen, L.-C. Nguyen, and D.-H. Nguyen, "A survey of IoT malware and detection methods based on static features," *ICT Exp.*, Apr. 2020.

[10] A. Azmoodeh, A. Dehghantanha, and K.-K. R. Choo, "Robust malware detection for Internet of (Battlefield) Things devices using deep eigenspace learning," *IEEE Trans. Sustain. Comput.*, vol. 4, no. 1, pp. 88–95, Jan.-Mar. 2018.

[11] H. HaddadPajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, "A deep recurrent neural network based approach for Internet of Things malware threat hunting," *Future Gener. Comput. Syst.*, vol. 85, pp. 88–96, Mar. 2018.

[12] H. Darabian, A. Dehghantanha, S. Hashemi, S. Homayoun, and K.-K. R. Choo, "An opcode-based technique for polymorphic Internet of Things malware detection," *Concurrency Comput.: Pract. Experience*, vol. 32, no. 6, p. e5173, Feb. 2020.

[13] "Virustotal," Accessed: Oct. 10, 2020. [Online]. Available: https://www.virustotal.com

[14] A. Marzano *et al.*, "The evolution of bashlite and mirai IoT botnets," in *Proc. IEEE Int. Symp. Comput. Commun.*, Jun. 2018, pp. 813–818.

[15] S. Edwards and I. Profetis, "Hajime: Analysis of a decentralized internet worm for IoT devices," *Rapidity Netw.*, vol. 16, 2016.

[16] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, "Measurement and analysis of Hajime, a peer-to-peer IoT Botnet," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Feb. 2019.

[17] M. J. Bohio, "Analyzing a backdoor/bot for the MIPS platform," 2015. Accessed: Oct. 10, 2020. [Online]. Available: https://www.sans.org/reading-room/whitepapers/malicious/analyzing-backdoor-bot-mips-platform-35902

[18] P. Kalnai and J. Horejsi, "DDoS trojan: A malicious concept that conquered the elf format," in *Proc. Virus Bull. Conf.*, 2015, pp. 62–70.

[19] "XORDDoS, kaiji botnet malware variants target exposed docker servers," Accessed: Oct. 10, 2020. [Online]. Available: https://blog.trendmicro.com/trendlabs-security-intelligence/xorddos-kaiji-botnet-malware-variants-target-exposed-docker-servers/

[20] Y. Ding and S. Zhu, "Malware detection based on deep learning algorithm," *Neural Comput. Appl.*, vol. 31, no. 2, pp. 461–472, Feb. 2019.

[21] B. Kang, S. Y. Yerima, K. McLaughlin, and S. Sezer, "N-opcode analysis for android malware classification and categorization," in *Proc. Cyber Secur.*, Jun. 2016, pp. 1–7.

[22] H. Alasmary *et al.*, "Analyzing and detecting emerging Internet of Things malware: A graph-based approach," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8977–8988, Jul. 2019.

[23] H.-T. Nguyen, Q.-D. Ngo, and V.-H. Le, "IoT botnet detection approach based on PSI graph and DGCNN classifier," in *Proc. Int. Conf. Inf. Commun. Signal Process.*, Sep. 2018, pp. 118–122.

[24] 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 Comput. Softw. Appl. Conf.*, vol. 2, Jul. 2018, pp. 664–669.

[25] 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.

[26] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proc. IEEE Symp. Secur. Privacy*, May 2000, pp. 38–49.

[27] B. Li, Y. Zhang, J. Yao, and T. Yin, "MDBA: Detecting malware based on bytes n-gram with association mining," in *Proc. Int. Conf. Telecommun.*, Apr. 2019, pp. 227–232.

[28] T. Ban, R. Isawa, S. Guo, D. Inoue, and K. Nakao, "Efficient malware packer identification using support vector machines with spectrum kernel," in *Proc. Asia Joint Conf. Inf. Secur.*, Jul. 2013, pp. 69–76.

[29] E. Foundation, "IoT commercial adoption survey 2019 results," 2019, Accessed: 2020-10-10.

[30] "binwalk," Accessed: Oct. 10, 2020. [Online]. Available: https://tools.kali.org/forensics/binwalk

[31] R. Isawa, T. Ban, S. Guo, D. Inoue, and K. Nakao, "An accurate packer identification method using support vector machine," *IEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vol. 97, no. 1, pp. 253–263, Jan. 2014.

[32] "Radare2," Accessed: Oct. 10, 2020. [Online]. Available: https://github.com/radareorg/radare2

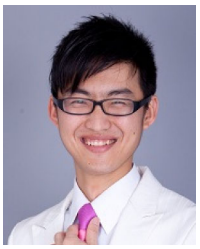
[33] "pwntools," Accessed: Oct. 10, 2020. [Online]. Available: hhttps://github.com/Gallopsled/pwntools

[34] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[35] L. McInnes, J. Healy, N. Saul, and L. Grossberger, "UMAP: Uniform manifold approximation and projection," *J. Open Source Softw.*, vol. 3, no. 29, p. 861, Sep. 2018.

TZU-LING WAN received the B.E. and M.E. degrees in computer science and information engineering from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 2018 and 2020, respectively. Her primary research interests include security for IoT and machine learning.

TAO BAN (Member, IEEE) received the B.S. degree from the Department of Automatic Control, Xi'an Jiaotong University, Xi'an, China, in 1999, the M.E. degree from the Department of Automation, Tsinghua University, Beijing, China, in 2003, and the Ph.D. degree from Kobe University, Kobe, Japan, in 2006. He is currently a Senior Researcher at Cybersecurity Research Institute, National Institute of Information and Communications Technology, Tokyo, Japan. His research interests include network security, malware analysis, machine learning and data mining for security, etc.



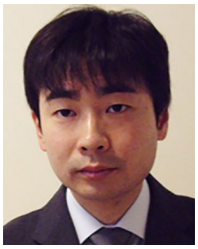
SHIN-MING CHENG (Member, IEEE) received the B.S. and Ph.D. degrees in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 2000 and 2007, respectively. He was a Postdoctoral Research Fellow at the Graduate Institute of Communication Engineering, National Taiwan University, from 2007 to 2012. Since 2012, he has been on the faculty in the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, where he is currently an Associate Professor. Since 2017, he has been with the Research Center for Information Technology Innovation, Academia Sinica, Taipei, where he is currently a joint Associate Research Fellow. His current research interests are secure mechanism design and security-related platform development in 4G/5G networks and IoT networks. Recently, he is investigating the robustness issue in machine learning. He received 2014 K. T. Li Young Researcher Award from ACM Taipei/Taiwan Chapter, IEEE PIMRC 2013 Best Paper Award, and CISC 2020 Best Paper Award.

YEN-TING LEE received the B.E. degree in computer science and information engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2018 and the M.E. degree from National Taiwan University of Science and Technology, Taipei, Taiwan, in 2020. His current research interests include security for IoT and machine learning.

BO SUN received the B.E. degree in science from Jilin University, Changchun, China in 2007, the M.E. degree in engineering from Yokohama National University, Yokohama, Japan in 2012, and the Ph.D. degree in engineering from Waseda University, Tokyo, Japan in 2018. He was at the University of Waseda as a Research Associate from 2016 to 2018, and a Researcher at the National Institute of Information and Communications Technology from 2018 to 2020. He is currently an Assistant Professor at the Saitama Institute of Technology, a collaborative Researcher at the National Institute of Information and Communications Technology, and a Visiting Researcher at the University of Waseda. His research interests include web security, mobile security, and offensive security.

RYOICHI ISAWA received the B.E. and M.E. degrees from the University of Tokushima, Tokushima, Japan at in 2004 and 2006, respectively, and the Ph.D. degree from Kobe University, Kobe, Japan, in 2012. He is currently a Senior Researcher at the National Institute of Information and Communications Technology (NICT), Japan. His current research interests include malware analysis, network security, and hardware security.

TAKESHI TAKAHASHI (Member, IEEE) received the Ph.D. degree in telecommunication from Waseda University, Tokyo, Japan, in 2005. He was at the Tampere University of Technology as a Researcher from 2002 to 2004, and Roland Berger, Ltd., as a Business Consultant, from 2005 to 2009. Since 2009, he has been with the National Institute of Information and Communications Technology, where he is currently a Research Manager. His primary research focus is cybersecurity. He is a member of the Association for Computing Machinery, and the Institute of Electronics, Information and Communication Engineers. He received several awards including Funai Information Technology Incentive Award and ITU Association Japan Incentive Award. He is a CISSP.



DAISUKE INOUE (Member, IEEE) received the B.E. and M.E. degrees in electrical and computer engineering and the Ph.D. degree in engineering from Yokohama National University, Yokohama, Japan in 1998, 2000, and 2003, respectively. He joined Communications Research Laboratory (CRL), Japan, in 2003. CRL was relaunched as National Institute of Information and Communications Technology (NICT) in 2004, where he is currently the Director of Cybersecurity Laboratory. He received several awards including the best paper award at the 2002 Symposium on Cryptography and Information Security (SCIS 2002), the commendation for science and technology by the Minister of MEXT, Japan, in 2009, the Good Design Award 2013, the Asia-Pacific Information Security Leadership Achievements (ISLA) 2014, the award for contribution to Industry-Academia-Government Collaboration by the Minister of MIC, Japan, in 2016, and the Maejima Hisoka Award, in 2018.