Cross Platform IoT-Malware Family Classification based on Printable Strings

Yen-Ting Lee*, Tao Ban‡, Tzu-Ling Wan*, Shin-Ming Cheng*†, Ryoichi Isawa‡, Takeshi Takahashi‡, Daisuke Inoue‡
*Department of Computer Science and Information Engineering,
National Taiwan University of Science and Technology, Taipei, Taiwan
{m10715076, m10715103, smcheng}@mail.ntust.edu.tw
†Research Center for Information Technology Innovation, Academia Sinica, Taipei, Taiwan
‡National Institute of Information and Communications Technology, Tokyo, Japan
{bantao, isawa, takeshi_takahashi, dai}@nict.go.jp

***Abstract*—In this era of rapid network development, Internet of Things (IoT) security considerations receive a lot of attention from both the research and commercial sectors. With limited computation resource, unfriendly interface, and poor software implementation, legacy IoT devices are vulnerable to many infamous malware attacks. Moreover, the heterogeneity of IoT platforms and the diversity of IoT malware make the detection and classification of IoT malware even more challenging. In this paper, we propose to use printable strings as an easy-to-get but effective cross-platform feature to identify IoT malware on different IoT platforms. The discriminating capability of these strings are verified using a set of machine learning algorithms on malware family classification across different platforms. The proposed scheme shows a 99% accuracy on a large scale IoT malware dataset consisted of 120K executable fils in executable and linkable format when the training and test are done on the same platform. Meanwhile, it also achieves a 96% accuracy when training is carried out on a few popular IoT platforms but test is done on different platforms. Efficient malware prevention and mitigation solutions can be enabled based on the proposed method to prevent and mitigate IoT malware damages across different platforms.**

***Index Terms*—Computer security, IoT malware, machine learning, malware analysis, static analysis**

I. Introduction

With its significant connectivity, flexibility, and applicability, the Internet of Things (IoT) has enjoyed great proliferation in recent years [1]. However, simple implementation, default password settings, and difficult-to-patch properties make IoT devices vulnerable to numerous malware attacks. Thus, implementing the security requirements of IoT devices is a critical and challenging issue.

Malware analysis, i.e. investigating the behavior of malware samples and classifying malware into families with similar features, has been studied broadly, and using various methods have been proposed to effectively prevent and mitigate malware on several platforms, e.g. the Windows and Android operating systems. IoT malware programs are designed to spread across multiple central processing unit (CPU) architectures; thus, cross-platform analysis of IoT malware is much more challenging compared to single-platform analysis of malware [2, 3].

Conventional techniques, e.g. pattern matching, are not effective in detecting unknown malware, a rapid shift to using machine-learning engines has been observed in antivirus solutions [4, 5].

Based on the type of features used in learning, existing malware analysis methods fall into two categories, i.e. dynamic and static analysis method. In dynamic analysis, malware binaries are executed in an emulated environment, e.g. a honeypot or sandbox; thus, the malware runtime behavior can be recorded and analyzed. In contrast, static analysis is performed on some version of the source code or object code. The logs files obtained from both dynamic and static analyses are subsequently taken as inputs to machine learning methods to generate prediction models for various purposes, e.g. malware detection and malware family classification.

IoT malware analysis can be pursued using the dynamic and static analysis paradigms, special attention has to be given to the inherent multi-platform nature of the problem. For IoT malware implemented on multiple CPU architectures, the development of emulation environments corresponding to different CPU architectures to support dynamic analysis is resource-intensive and sometimes suffers from problems such as software incompatibility [6]. In addition, crafty malware can be implemented with mechanisms to detect emulation environments; thus, such malware can evade dynamic analysis [5] methods. Therefore, static analysis is currently considered much more efficient for IoT malware when implementation principles, e.g. simplicity and straightforwardness, are valid. However, not all static features are effective for cross-platform analysis. For example, operation codes (opcodes) defined in instruction sets on a specific CPU architecture differ completely from those defined on other CPU architectures. Another example of architecture-dependent structure is the executable and linkable format (ELF) header. Although the ELF header is a common standard format defined for Unix and Unix-like systems, values in the header fields are heavily dependent on the platform on which the files are complied. In this sense, opcodes and ELF headers are architecture-dependent and cannot be used to facilitate cross-platform IoT malware

978-1-6654-0392-4/20/$31.00 ©2020 IEEE
DOI 10.1109/TrustCom50675.2020.00106

analysis.

In this paper, we explore using static features to develop an advanced malware analyzer that can generalize knowledge learned from malware on one CPU architecture to predict malware on other CPU architectures. We propose the use of *printable strings* as input features to build machine learning models for malware analysis. Typically, it comprises comprehensible texts, e.g. plain text in code comments, function names created by authors, or application programming interface (API) names generated by the compiler. Printable strings have the following advantages compared to other commonly used features, e.g. opcodes, ELF headers, and byte sequences.

- High accessibility: Printable strings can be extracted directly from malware binaries with little effort; thus, this is a cost-effective approach for IoT devices with limited resources. This a strong advantage over other features, e.g. opcodes, that require advanced reverse engineering tools for feature extraction.
- High intelligibility: Readable content, e.g. plain texts and function names, are intelligible for users without malware reverse engineering or digital forensics backgrounds. In addition, it facilitate better explanations of the prediction results obtained from the analysis.
- Cross-platform generalization ability: Printable strings include essential identifying information that is closely related to the source code; therefore, they can capture common characteristics of malware samples from the same family compiled on different CPU architectures.

In this study, we collected a large IoT malware dataset and performed a series of numerical investigations to evaluate the generalization performance of the proposed cross-platform malware analysis method. The malware set downloaded from VirusTotal [7] comprised 122,504 recent malware samples compiled on seven distinct CPU architectures, i.e. ARM, MIPS, X86, X86-64, PowerPC, SPARC, and UNKNOWN[1]. To verify the cross-platform performance of the proposed method, a training dataset comprised malware samples on the top-three major CPU architectures, and the test dataset comprised IoT malware samples from CPU architectures not included in the training samples. Specifically, ARM, MIPS, and X86 samples were selected for the training set, and SPARC, X86-64, PowerPC, and UNKNOWN samples were used for testing. To remove noisy and redundant strings that were irrelevant to the learning, we designed a feature selection scheme that reduced feature dimensionality from 14 million to approximately 2K, which significantly reduced training time and increased prediction performance.

Among the three evaluated classifiers (i.e. the random forest (RF), K nearest neighbors (KNN), and support vector machine (SVM) classifiers), the SVM classifier demonstrated the best performance and obtained an accuracy rate of up to 99.36% on the training set when evaluated by 10-fold cross-validation. The accuracy rate obtained when using the classifiers on the testing sets for other CPU architectures was up to an average of 98%, while the methods opcode and ELF header-based methods showed an average accuracy of 32.78% and 81.76%, respectively. These results verify the efficacy of using printable strings to obtain discriminating information to facilitate malware classification on known platforms and generalizing to those on unknown platforms.

The remainder of this paper is organized as follows. Section II reviews work related to malware analysis. Section III defines the cross-platform analysis problem. Section IV presents the proposed cross-platform IoT malware analysis method. Section V evaluates the performance of the proposed method and discusses the experimental results. Finally, Section VI concludes the paper.

II. Related Work

In this section, we review studies related to the detection and classification of malware using machine learning methods. According to the method used to analyze malware, they are categorized into two paradigms, i.e. static analysis and dynamic analysis [8].

A. Dynamic analysis-based approaches

In dynamic analysis, program binaries are executed in an emulated environment; thus, their runtime behaviors can be recorded and analyzed. Among the various system levels at which dynamic analyses can be performed, system calls provide a comprehensive interface between software and the Linux kernel, and such analyses are prominent in the literature. For example, Xiao et al. [9] took the semantic information in the system call sequences as natural language, and they extracted system call sequences as a sentence and constructed a classifier based on long short term memory (LSTM) neural networks. In addition, Shobana et al. [10] extracted system calls from ELF files and preprocessed them using the N-gram model. They then used the N-gram representation to train a recurrent neural network (RNN) for malware detection. Dynamic analysis can also be pursued using network traffic captured during software runtime. For example, Wang et al. [11] collected the network traffic generated by Android mobile applications and transmitted it to a central server for analysis. On the server side, numerical features were extracted from the traffic and input to machine learning engines for malware detection.

Dynamic analysis provides an effective way to understand the attacking behaviors of malware and differentiate them from benign software. Depending on the common system call names shared among different CPU architectures, cross-platform dynamic analysis can be implemented to various extents. Nonetheless, the heterogeneity of IoT devices' CPU architectures may lead to unforeseeable complexity in developing a consistent virtual environment that works for multiple CPU architectures [5]. In addition, the resource consumption of deploying such a virtual environment is considered infeasible on resource-constrained IoT devices [6].

[1]Malware samples with unidentified CPU architecture information are categorized as UNKNOWN.

B. Static analysis-based malware detection

In contrast to dynamic analysis, static analysis is performed without executing the programs. Such analyses are commonly performed on some version of the source or object code. Therefore, reduced resource consumption and improved modelling efficiency can be expected with static analysis.

For example, Darabian et al. [12] counted the numbers of repeated occurrences of opcodes to detect malware, and they found that the numbers of repetitions of certain opcodes in malware samples are greater than those in benignware samples. In addition, Haddadpajouh et al. [13] extracted the opcode sequence and selected useful features based on the *information gain* scores of the opcodes. Then, they applied an LSTM network to these features for malware detection. Azmoodeh et al. [14] and Dovom et al. [15] obtained the opcode sequence with program execution order using a control flow graph (CFG). They adopted a deep learning algorithm and a fuzzy pattern tree to detect IoT malware. Yuxin et al. [16] also used a CFG to obtain the opcode sequence with the program execution sequence. They then constructed an N-gram representation of these sequences and applied a deep belief network (DBN) for malware detection. Kang et al. [17] extracted N-grams from opcode sequences and encoded them into binary codes, and the N-gram representation was then used to train a classifier for malware detection.

The opcode can effectively detect and classify malware; however, due to differences in instruction sets under different CPU architectures, opcodes are limited to the same architecture for generalization. Another type of easily extracted feature is from the ELF format. For example, Shahzad et al. [18] extracted 383 structural features from ELF files, including section headers, symbols section, and program headers. Then, they used forensic analysis to sort and select effective features. However, ELF fields are typically set to predetermined values depending on the CPU architecture an executable is compiled; thus, classifiers based on ELF-related features tend to demonstrate reduced generalization performance in cross-platform analysis.

It is also common to analyze malware using the feature structure of a directed graph. For example, Alasmary et al. [19] found that IoT malware contains fewer nodes and edges than Android malware in the binary file. Alasmary et al. [20] also used *Radare2* to generate a CFG of the sample and calculated the CFG properties, e.g. density, the shortest path, and the numbers of edges and nodes. Then, machine learning algorithms were used to classify samples as malware or benignware based on the CFG features. Malware detection via CFG features focuses on only the structure of the code block and generalizes well to samples from different CPU architectures. However, calculating the graph properties tends to be time consuming for large files, which can lead to reduced detection efficiency.

In addition, Islam et al. [21] extracted printable strings and API calls from Windows PE to integrate static and dynamic analyses. Here, the static features used to train the classification models included function length frequency, frequency of printable strings, e.g. API calls, and a one-hot encoding representing the presence of printable strings. Incorporating static features in learning resulted in a significant improvement in malware detection (from 90.398% to 97.055%) on an integrated dataset.

Alhanahnah et al. [3] used N-gram extract meaningful printable string sequences from the executables. They performed malware detection based on high-level statistical features from the assembly codes, including: total number of functions, total number of instructions, number of redirect instructions, number of arithmetic instructions, number of logical instructions, number of transfer instructions, number of segments, and number of call instructions.

ba executables, and they performed malware detection based on high-level statistical features from the assembly codes, including total number of functions, total number of instructions, number of redirect instructions, number of arithmetic instructions, number of logical instructions, number of transfer instructions, number of segments, and number of call instructions.

III. Cross-platform Analysis: Problem Definition

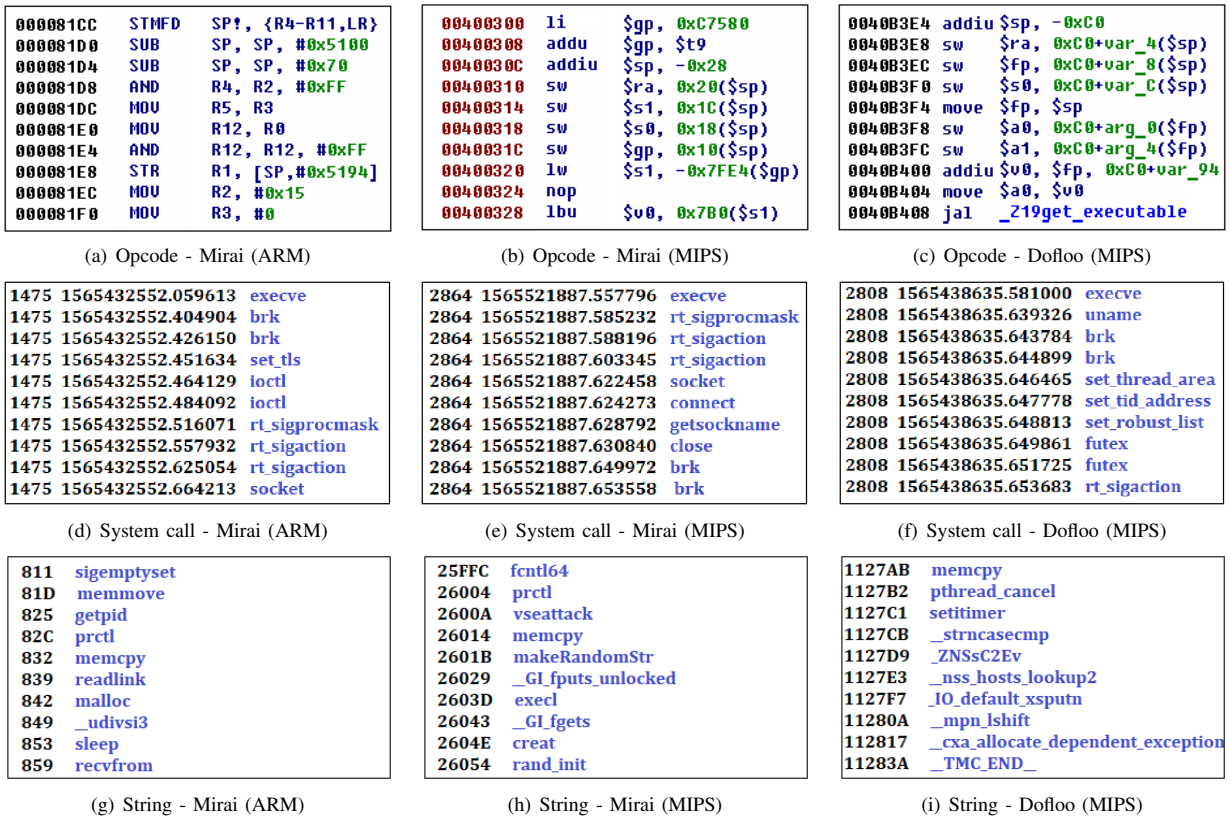
Here, we describe our motivation and philosophy behind performing IoT-malware analysis across different CPU architectures. The Linux operating system is the most popular for IoT devices [22]. Most IoT malware arrive on IoT devices in the executable and linkable format (ELF), which is the standard binary format on Unix-like operating systems. By design, the ELF is flexible, extensible, and cross-platform, and this flexibility allows it to be adopted by many different operating systems on many different hardware platforms.

A. Motivation

Unlike conventional malware on Windows and Android devices, which are confined to a limited number of CPU architectures, IoT devices have much more diverse CPU architectures, and IoT malware can spread among heterogeneous devices [3]. These differences in CPU architectures cause the same malicious behavior to be presented with different characteristics. In addition, the distribution of malware over different CPU architectures is extremely uneven. Consider the Dofloo distribution example shown in Table I. Most malware samples are collected on the X86 and ARM architectures, with fewer samples collected on the MIPS and X86-64 architectures. There are no Dofloo samples collected on other CPU architectures. In this case, an effective malware analyzer must learn the identifying information from Dofloo samples on available platforms and generalizes this information to detect malware on other CPU platforms.

B. Effective features

In the previous section, assorted features obtained from static and dynamic analyses were surveyed in the malware analysis context. Many of them do not have strong potential to facilitate cross-platform classification, which reduces



(a) Opcode - Mirai (ARM) (b) Opcode - Mirai (MIPS) (c) Opcode - Dofloo (MIPS)

(d) System call - Mirai (ARM) (e) System call - Mirai (MIPS) (f) System call - Dofloo (MIPS)

(g) String - Mirai (ARM) (h) String - Mirai (MIPS) (i) String - Dofloo (MIPS)

Fig. 1. Comparison of static features for different platform and malware families

classification performance for IoT malware in heterogeneous environments. Fig. 1 shows examples of different raw features extracted from the ELF files of different CPU architectures and malware families. Consider the example opcode sequences shown in the first row of Fig. 1. Here, while the opcode sequence of the Mirai sample in (b) shows strong similarity with the Dofloo sample in (c), it appears to be completely different from the Mirai sample in (a) because different CPU architectures tend to adopt completely different instruction sets. In this sense, due to the difference in representing features, discriminating knowledge learned for a specific CPU architecture cannot be generated for CPU architectures with different instruction sets.

In the proposed IoT malware analysis method, malware binaries in ELF format are taken in as input. As long as no intended obfuscation is adopted when creating ELF files, which is the current situation for IoT malware, the binary file contains identifying information for the functionality carried out by the program. Although ELF is defined to facilitate machines to perform predetermined instructions in the programs, information contained in the file can be partially read by humans. Fig. 2 shows the hex dump of the *bash* command in ELF on the ARM64 platform[2]. The left panel of the hex dump shows the hexadecimal ASCII codes of the bytes in the ELF file. In the right panel, only the printable strings are interpreted and printed for ease of reading. Here, the color boxes highlight three major types of readable contents, i.e. API names (green), garbled texts (red), and plain texts (blue). Examples of printable string segments collected from malware samples on different IoT CPU architectures are shown at the bottom in Fig. 1. For the two Mirai samples (g) and (h) collected from ARM and MIPS, respectively, the displayed segments comprise system calls that appear to be from the same source. In contrast, the system calls shown in the Dofloo sample in (i) appear to be drawn from a different set.

[2]This program is available at: https://github.com/JonathanSalwan/binary-samples/blob/master/elf-Linux-ARM64-bash.

C. Problem definition

We have established the following requirements on the proposed cross-platform IoT malware analysis scheme.

- The conversion from software programs to a unified format as input to the model can be performed regardless of the CPU architecture programs are compiled on.
- The conversion from a test software program to input to the prediction model can be performed regardless of the CPU architecture on which the training samples are compiled.

The first requirement guarantees that any ELF file collected from IoT devices can serves as a training sample to build the

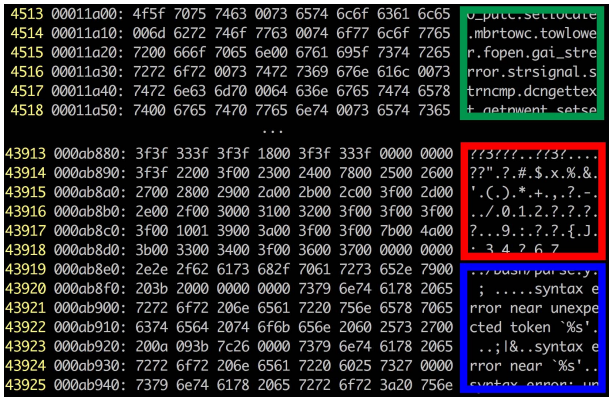


Fig. 2. Printable strings in ELF binaries

model or a test sample to evaluate the model. This requirement also implies a unified model. The second requirement guarantees that generalization can be extended to previously unknown platforms.

IV. Methodology

In this section, we describe the proposed IoT malware analysis method based on printable strings extracted from binaries. As shown in Fig. 3, a given malware dataset is analyzed following four steps, i.e. feature extraction, feature selection, model training, and model evaluation. In the feature extraction step, we extract printable strings from malware samples and encode them as numerical vectors. In the feature selection step, we introduce feature ranking criteria to estimate the significance of features to obtain superior classifiers with reduced feature dimension and faster training time. In the model training step, the feature vectors are input to classification algorithms to construct prediction models. Finally, in the model evaluation step, the generalization performance of the models is evaluated using the test samples.

A. Feature extraction

The proposed method analyzes the ELF binaries of IoT malware and employs the trained classifier to each platform for prediction. We extract the information from the binary files and formulate the malware samples as input vectors for machine learning models. We consider two feature vectors, i.e. a *string length frequency (SLF)* vector and a *printable string information (PSI)* vector. In addition, we compare our method with the static features presented in [21].

1) String length frequency vectors: The SLF vector contains information of the number of strings in different length ranges. We calculated the length of all strings extracted from malware and calculated how many strings were contained in each bin with a predefined length range. Finally, we take the values of all bins to form the SLF vector (Fig. 4). Here, we used 50 bins, and the jth bin was defined to contain strings with a length range between $e^{(j-1)\ln(L)/50}$ and $e^{(j)\ln(L)/50}$, where L is the maximum string length in the complete dataset. L was set to 74,076 in our experiment.

2) Printable string information vectors: First, we created a global list containing all printable strings extracted from the X86, ARM, and MIPS binaries in the dataset. We then determined whether the strings in the global list exist in each ELF. A file x_i with printable strings is encoded as a binary vector $X_i \in \{0,1\}^D$, where D is the length of the global list, $x_{i,j} = 1$ indicates that string i shows up in the jth malware, and $x_{i,j} = 0$ indicates the absence of string i in the jth malware.

B. Feature selection

Initially, we extracted 14,530,806 printable strings as features; however, some of these features may contain noisy information and lead to degenerated classifier performance. By calculating the *document frequency (DF)* of each string, we excluded strings with low frequency, e.g. those with DF less than 10. In addition, we did not consider strings with special characters because such strings may be generated by the compiler and may have negative effect to cross-platform capability. To reduce the redundancy in co-occurring strings, we found out string sets that have strong co-occurrence and kept one of them in our feature set. As a result, the feature dimension was reduced to 212,146 before the application of feature selection methods to further reduce dimensionality and improve cross-platform capabilities.

In the first method, we adopted the recursive feature elimination (RFE) [23] algorithm for feature selection. The RFE algorithm calculated each feature's score by training a support vector machine (SVM) model, and removed features with low scores. Then, the RFE algorithms repeated the iteration with the remaining features until the specified number of features was obtained.

In the second method, cross-platform capability was considered, where we obtained cross-platform information from the distribution of DF. Here, we used information entropy to measure the uncertainty in the ith string's distribution on different CPU architectures.

Let

$$y_i = \{y_{i,j} | j = 1, \ldots, M\}, \tag{1}$$

be the set of DF values of the i-th string on M CPU architectures. Information entropy, $H(y_i)$, which indicates the average degree of information in y_i, is computed as

$$H(y_i) = -\sum_{j=1}^{M} P(y_{i,j}) log P(y_{i,j}), \tag{2}$$

where $P(y_{i,j})$ is the probability when $y_i = y_{i,j}$. Then we define a compound score, namely *DFrank*, for the ith string as follows:

$$DFrank(y_i) = H(y_i) \times \frac{df_{i,k}}{df_i}, \tag{3}$$

where $df_{i,k}$ is the DF value of string i in the kth class, and df_i is the sum of all DF values of string i. The first multiplier in DFrank, i.e. the information entropy, rewards strings that have

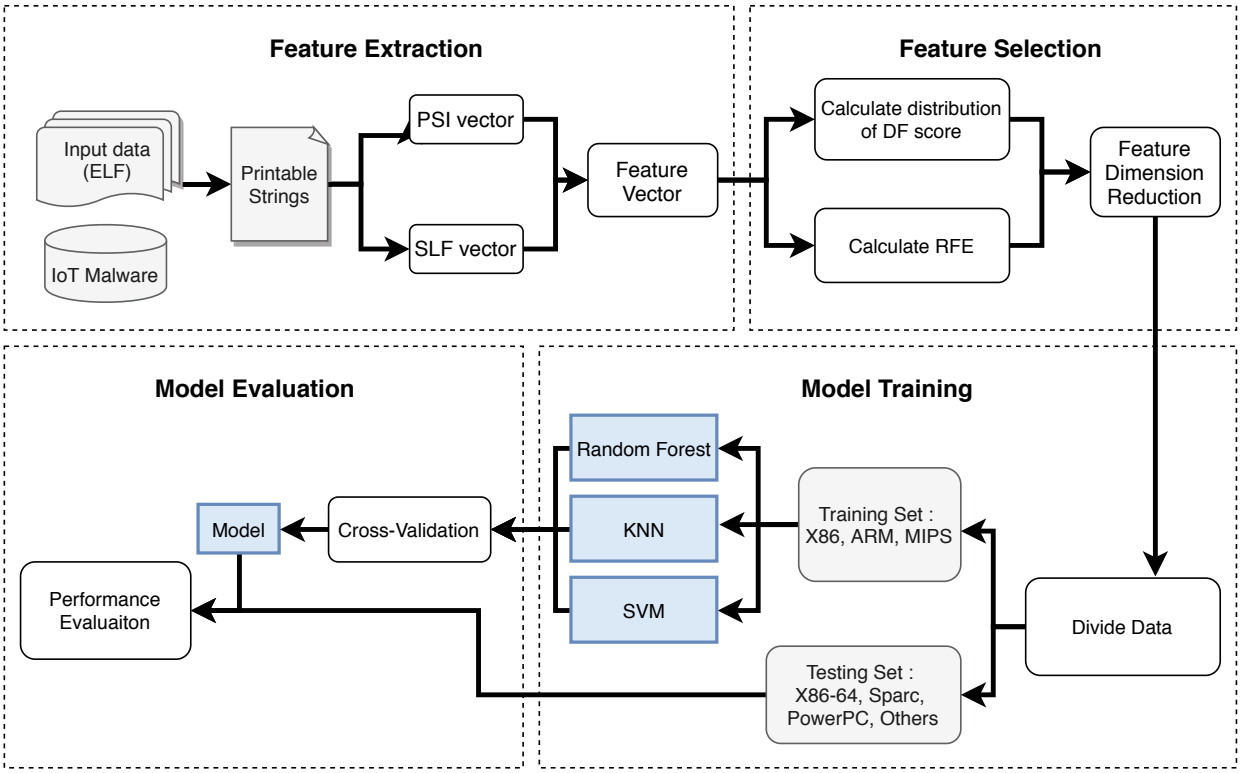


Fig. 3. System model

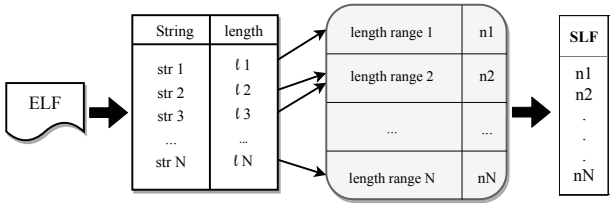


Fig. 4. Distribution among string length frequency bins

high generalization capability across different CPU architectures. The second multiplier in DFrank rewards strings with much higher DF values in the kth class than in the other classes and therefore can be considered as identifying features of the kth class. By taking strings with high DFrank scores from each family, we can obtain features with high discriminating information and better cross-platform capability.

Based on RFE and DFrank, we tried different strategies to combine them for improving the feature selection results. The settings we adopted in our experiments are discussed in V-C.

C. Machine-learning methods

Classifiers trained by machine learning methods can find rules automatically by analyzing data, which is beneficial to analyzing unknown malware in a variety of malware environments. Here, we selected three commonly used classifiers, i.e. the *random forest(RF)*, k nearest neighbors *KNN*, and *SVM* classifiers. To comply with the cross-platform analysis problems defined in Section III, we considered malware samples collected from the three most common CPU architectures, i.e. the X86, ARM, and MIPS architectures, as the training set. Assuming that malware on other platforms is unknown for training, we cannot take account of string features other than those find in the training samples. Using the aforementioned feature extraction and selection methods, the strings were converted to a feature vector and input to the machine learning algorithms. To evaluate the performance of the cross-platform analysis, we used 10-fold cross validation, and we evaluate the testing set after each fold was trained. The training effectiveness and cross-platform analysis results are based on the average of 10-fold cross validation. We use Scikit-Learn [24] to implement the algorithms described above.

V. Performance Evaluation

In this section, we evaluate the performance of the classifiers trained on the IoT malware dataset. First, we introduce the experimental dataset. Then, we evaluate the proposed method before and after feature selection, and compare the performance of different machine learning algorithms. Finally, the proposed method is applied to compare performance with classifiers trained on two types of static features other than printable strings.

TABLE I

CPU ARCHITECTURE DISTRIBUTION IN EACH LABEL

	Training set			Testing set			
	X86	MIPS	ARM	SPARC	X86-64	PPC	UNKNOWN
Mirai	15806	10089	18659	3655	3385	5033	7267
Tsunami	844	318	382	42	190	90	165
Hajime	0	158	321	0	0	0	0
Dofloo	404	83	979	0	80	0	0
Bashlite	6481	6917	10212	2359	2977	2709	3950
Xorddos	472	0	0	0	3	0	0
Android	1073	357	2857	0	692	1	5

A. Dataset

We collected 122,504 malware ELF files from VirusTotal [7], which were divided into seven malware families according to antivirus software reports, i.e. Mirai, Tsunami (Kaiten), Hajime, Dofloo, Bashlite (Gafgyt), Xorddos, and Android. In addition, these malware ELF files were further divided into X86, MIPS, ARM, SPARC, X86-64, PPC, and unknown type according to the CPU architecture. Table I shows the distribution of malware families and CPU architectures.

B. Evaluation metrics

In these evaluations, we adopted common evaluation metrics (e.g. accuracy, precision, and false positive rate) to investigate the performance of the proposed method. These metrics are defined based on the following intermediate measures.

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

Accuracy is the percentage of correctly classified test samples:

$$Accuracy = \frac{TP + TN}{n}. \tag{4}$$

Precision represents the probability that predicted positives are classified correctly:

$$Precision = \frac{TP}{TP + FP}. \tag{5}$$

Recall is the probability that samples in a given class are classified correctly:

$$Recall = \frac{TP}{TP + FN}. \tag{6}$$

F1-measure is the weighted average of precision and recall:

$$F1measure = \frac{2 * (Recall * Precision)}{Recall + Precision}. \tag{7}$$

C. Performance evaluation

1) Performance before feature selection: We referenced the analysis method through printable strings in [21], and applied it to the classification of IoT malware. Table II shows the classification performance before feature selection. With this method, 97.95% cross validation accuracy was obtained on the dataset before performing feature selection. However, the dimension of the feature vector was as high as 14,530,806, and training required excessive time; thus, we adopted feature selection to reduce training time.

TABLE II

CROSS-PLATFORM PERFORMANCE BEFORE FEATURE SELECTION

	CPU architecture	Algorithm	Accuracy(%)	F1-measure(%)
Training Set	X86 + ARM + MIPS	RF	97.95	97.94
		KNN	97.12	97.11
		SVM	97.45	97.44
Testing Set	SPARC	RF	98.00	98.00
		KNN	94.83	95.94
		SVM	98.04	98.05
	X86-64	RF	93.88	93.72
		KNN	89.00	88.54
		SVM	92.72	92.31
	PPC	RF	96.95	96.93
		KNN	87.50	88.79
		SVM	96.86	96.85
	Unknown	RF	98.59	98.52
		KNN	69.28	74.73
		SVM	97.77	98.30

2) Feature selection result: Fig. 5 shows the results of our cross-platform performance evaluation for each experiment setting, including the accuracy of known and unknown malware classification, feature dimension, and how many files can obtain the selected features. The feature selection setting corresponding to each line in the table is summarized follows.

(a) Removing redundant and irrelevant features.
(b) Selecting the top-1000 strings with higher scores using RFE.
(c) Using DFrank scores to rank the most import strings for each malware family with high ranked strings selected.
(d) Combining strings selected by methods (b) and (c).
(e) Applying RFE to results in (d) and selecting high ranking features.
(f) Applying setting (c) to select 20,000 features from each family and performing RFE ranking.

In setting (a), we deleted unimportant features and reduced the feature dimension from 14 million to 210K, with training cost reduced significantly. The classifier trained on the remaining string features achieved an average accuracy of 98.37%, and it maintained an average accuracy of 97.2% when predicting malware in the testing set.

In setting (b), RFE was applied to obtain top ranking string features, after the feature selection, the accuracy rate rose to 98.59%. This setting yielded 97.87% prediction accuracy on unknown platforms, which is a remarkable improvement as compared with setting (a). Note, RFE did not take account of cross-platform capabilities when selecting features.

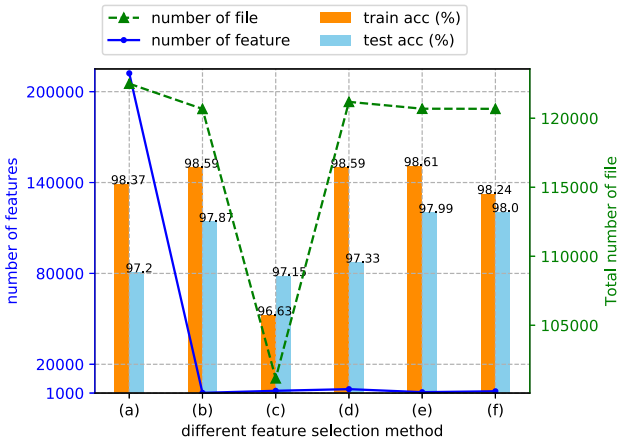


Fig. 5. Performance comparison of feature selection settings

TABLE III
COMPARISON OF TRAINING COMPLEXITY BEFORE AND AFTER FEATURE SELECTION

	Before feature selection			After feature selection		
	Feature Dimension	Training Time (s)	Accuracy (%)	Feature Dimension	Training Time (s)	Accuracy (%)
RF	14,530,806	100,557	97.95	2,193	140	**98.34**
KNN		4,391	97.12		340	**98.03**
SVM		85,819	97.45		46	**98.36**

In setting (c), we selected string features with more cross-platform capability by considering the cross-platform characteristics; however, approximately 20K malware samples did not contain the string selected by this method, which resulted in reduced accuracy rates on the training and testing sets as 96.63% and 97.15%, respectively.

In setting (d), by combining the strings of from (b) and (c), we obtained strings that were easy to extract and had cross-platform capability. Here, malware classification accuracy rates of 98.59% and 97.33% were obtained on the training and testing sets.

Then, with setting (e), we performed additional RFE feature selection for the features selected by setting (d). Here, cross-platform performance was improved to 97.99%.

Finally, we used setting (c) to select 20,000 features and applied RFE to select the top 2,000 features. Average accuracy rates of 98.24% and 98.00% were obtained on the training and test sets, respectively.

As shown in Fig. 5, setting (f) demonstrates the highest cross-platform classification accuracy among all feature selection settings. Table III shows that the dimensionality of our feature vector was up to 14 million before feature selection, which caused excessive training time. Feature selection reduced the dimensionality of the vectors to 2,361 via feature selection, resulting in greatly reduced training time and improved accuracy.

TABLE IV
CROSS-PLATFORM CLASSIFICATION RESULTS OF DIFFERENT ML METHODS BASED ON PRINTABLE STRING

	CPU architecture	Algorithm	Accuracy(%)	F1-measure(%)
Training Set	X86 + ARM + MIPS	RF	98.34	98.33
		KNN	98.03	98.01
		SVM	98.36	98.34
		NB	97.03	97.01
		MLP	**98.35**	98.34
Testing Set	SPARC	RF	**98.40**	98.41
		KNN	98.17	98.18
		SVM	97.95	97.96
		NB	96.90	96.92
		MLP	98.37	98.37
	X86-64	RF	96.54	96.58
		KNN	94.80	94.59
		SVM	**96.83**	96.87
		NB	95.39	95.46
		MLP	96.74	96.78
	PPC	RF	98.49	98.51
		KNN	98.36	98.38
		SVM	98.78	98.78
		NB	96.68	96.83
		MLP	**99.28**	99.35
	Unknown	RF	**99.49**	99.49
		KNN	99.15	99.18
		SVM	99.04	99.13
		NB	95.48	95.45
		MLP	98.52	98.53

3) Comparison of common machine learning algorithms: In addition to selecting important features, choosing appropriate machine learning algorithms also greatly affect the generalization performance. Here, we compare five common machine learning algorithms, namely, RF, KNN, SVM, naive Bayes (NB), and multilayer perceptron (MLP). The results are shown in Table IV. On the training set, with the exception of NB which obtained an accuracy rate of only 97%, these algorithms reached an accuracy rate of 98%. Moreover, all algorithms performed fairly well on classifying malware with unknown CPU architectures. Among them, RF demonstrated the best average accuracy, and NB showed inferior performance compared to other algorithms.

D. Comparison of static features

In this experiment, we compare the cross-platform performance of the proposed method with classifiers built on features extracted from two other static analysis methods. We trained the classifiers based on printable string, opcode, and the ELF header with 10-fold cross-validation and evaluated their performance. The N-gram representation can take account of the continuity of the opcode sequence, it has be widely adopted in opcode-based analysis [25, 26]. On the other hand, ELF header is proofed to have excellent generalization performance for malware family classification, despite of the fact that it relies heavily on the CPU architecture. Note that feature vector extraction for opcode [17] and ELF header [18] are described in Section II.

TABLE V
CROSS-PLATFORM CLASSIFICATION RESULTS BASED ON DIFFERENT STATIC FEATURES

	CPU architecture	Algorithm	String		Opcode [17]		ELF header [18]	
			Accuracy(%)	F1-measure(%)	Accuracy(%)	F1-measure(%)	Accuracy(%)	F1-measure(%)
Training Set	X86 + ARM + MIPS	RF	98.34	98.33	96.23	96.20	98.80	98.79
		KNN	98.03	98.01	95.83	95.81	94.01	93.96
		SVM	98.36	98.34	96.09	96.05	94.35	94.15
Testing Set	SPARC	RF	98.40	98.41	53.66	43.68	91.88	91.65
		KNN	98.17	98.18	0.46	0.91	78.56	79.29
		SVM	97.95	97.96	15.78	16.43	93.05	92.78
	X86-64	RF	96.54	96.58	85.38	84.78	76.54	77.33
		KNN	94.80	94.59	79.79	80.95	71.71	72.22
		SVM	96.83	96.87	67.82	67.69	86.94	85.32
	PowerPC	RF	98.49	98.51	7.97	8.10	85.77	85.29
		KNN	98.36	98.38	11.84	16.76	80.24	79.84
		SVM	98.78	98.78	35.57	45.87	93.79	93.22
	Unknown	RF	99.49	99.49	25.50	29.20	76.41	72.18
		KNN	99.15	99.18	0.23	0.46	65.32	58.72
		SVM	99.04	99.13	9.41	13.89	80.93	78.89

The performance comparison result is shown in Table V. As shown in the table, on the training set, classifiers based on printable string features achieved optimal accuracy of 98.34%, while classifiers based on opcode and ELF header features obtained optimal accuracy rates as 96.23% and 98.80%, respectively. However, on the testing set, the performance of the classifier trained based on opcode and ELF header features degenerated significantly, obtaining average accuracy rates of 32.78% and 81.76%, respectively. In contrast, the printable-string-based classifier maintained an average accuracy of 98%. The result indicates that, when predicting malware of a new CPU architecture, printable string features demonstrate good cross-platform classification capabilities. However, the feature vectors created from opcodes and ELF headers changed significantly due to the different instructions and structures in each architecture, resulting in a significant degeneration in prediction accuracy. The classifier trained using printable strings can make predictions on various platforms, thereby making IoT malware easy to detect on heterogeneous devices.

VI. CONCLUSION

In this paper, we have proposed a method for effective cross-platform analysis of IoT malware. We collected ELF files of IoT malware, extracted printable strings from ELF binaries, and obtained useful features through different feature selection methods. We classified the malware family using the RF, KNN, and SVM algorithms. We obtained an accuracy rate of 98% in the training set and maintained an accuracy rate of 94% to 99% in our cross-platform analyses. The experimental results demonstrate that the proposed method can effectively differentiate malware across different platforms, and the easy-to-extract string allows us to develop a lightweight system for classification tasks.

REFERENCES

[1] M. A. Al-Garadi, A. Mohamed, A. Al-Ali, X. Du, I. Ali, and M. Guizani, "A survey of machine and deep learning methods for Internet of Things (IoT) security," *IEEE Communications Surveys & Tutorials*, Apr. 2020.

[2] A. Costin and J. Zaddach, "IoT malware: Comprehensive survey, analysis framework and case studies," *BlackHat USA*, 2018.

[3] M. Alhanahnah, Q. Lin, Q. Yan, N. Zhang, and Z. Chen, "Efficient signature generation for classifying cross-architecture IoT malware," in *Proc. IEEE CNS 2018*, Jun. 2018, pp. 1–9.

[4] R. Ito and M. Mimura, "Detecting unknown malware from ASCII strings with natural language processing techniques," in *Proc. 14th AsiaJCIS 2019*, Aug. 2019, pp. 1–8.

[5] 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 Express*, Apr. 2020.

[6] S.-M. Cheng, T. Ban, J.-W. Huang, B.-K. Hong, and D. Inoue, "ELF analyzer demo: Online identification for IoT malwares with multiple hardware architectures," in *Proc. IEEE S&P workshop 2020*, 2020.

[7] "Virustotal," https://www.virustotal.com.

[8] T. Ban, R. Isawa, S. Huang, K. Yoshioka, and D. Inoue, "A cross-platform study on emerging malicious programs targeting iot devices," *IEICE Trans. Inf. Syst.*, vol. 102-D, no. 9, pp. 1683–1685, 2019.

[9] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and LSTM," *Multimedia Tools and Applications*, pp. 3979–3999, Feb. 2019.

[10] M. Shobana and S. Poonkuzhali, "A novel approach to detect IoT malware by system calls using deep learning

techniques," in *Proc. ICITIIT 2020*, Feb. 2020, pp. 1–5.
[11] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, "A mobile malware detection method using behavior features in network traffic," *Journal of Network and Computer Applications*, vol. 133, pp. 15–25, May 2019.
[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 and Computation: Practice and Experience*, vol. 32, no. 6, p. e5173, Feb. 2020.
[13] 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 Generation Computer Systems*, vol. 85, pp. 88–96, Mar. 2018.
[14] A. Azmoodeh, A. Dehghantanha, and K.-K. R. Choo, "Robust malware detection for Internet of (Battlefield) Things devices using deep eigenspace learning," *IEEE transactions on sustainable computing*, vol. 4, no. 1, pp. 88–95, Feb. 2018.
[15] 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," *Journal of Systems Architecture*, vol. 97, pp. 1–7, Aug. 2019.
[16] D. Yuxin and Z. Siyi, "Malware detection based on deep learning algorithm," *Neural Computing and Applications*, vol. 31, no. 2, pp. 461–472, Feb. 2019.
[17] B. Kang, S. Y. Yerima, K. McLaughlin, and S. Sezer, "N-opcode analysis for android malware classification and categorization," in *Proc. IEEE Cyber Security 2016*, Jun. 2016, pp. 1–7.
[18] F. Shahzad and M. Farooq, "ELF-miner: Using structural knowledge and data mining methods to detect new (linux) malicious executables," *Knowledge and information systems*, vol. 30, no. 3, pp. 589–612, Mar. 2012.
[19] H. Alasmary, A. Anwar, J. Park, J. Choi, D. Nyang, and A. Mohaisen, "Graph-based comparison of IoT and android malware," in *Proc. Computational Social Networks 2018*, Nov. 2018, pp. 259–272.
[20] 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, Jul. 2019.
[21] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 646–656, Mar. 2013.
[22] E. Foundation, "Iot commercial adoption survey 2019 results," 2019.
[23] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, vol. 46, no. 1-3, pp. 389–422, Jan. 2002.
[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[25] T.-L. Wan, T. Ban, Y.-T. Lee, S.-M. Cheng, R. Isawa, T. Takahashi, and D. Inoue, "IoT-malware detection based on byte sequences of executable files," in *Proc. IEEE ASIAJCIS 2020*, 2020.
[26] T. L. Wan, T. Ban, S. M. Cheng, Y. T. Lee, B. Sun, R. Isawa, T. Takahashi, and D. Inoue, "Efficient detection and classification of internet-of-things malware based on byte sequences from executable files," *IEEE Open Journal of the Computer Society*, pp. 1–1, 2020.